An Application of Object-Functional Programming to Defence Modelling

Gareth David Toomey

May 2019

This thesis is submitted in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy of the University of Portsmouth.

Release Conditions

Whilst registered as a candidate for the above degree, I have not been registered for any other research award. The results and conclusions embodied in this thesis are the work of the named candidate and have not been submitted for any other academic award.

Word Count: 37,202

Acknowledgements

Firstly, I would like to thank my supervisor, Doctor Rich Boakes for his guidance and support throughout this research.

I would like to thank Rick Ansell, Hannah Higgins, Tim Chapman and all those who were part of the GAMOV team over the years. Without your efforts, GAMOV would not have been possible.

I would like to thank Paul Glover, Jon Hardy and Doctor Simon Collander-Brown for their support and review of this thesis within Dstl.

Lastly, I would like to thank my Mum: Elaine and Dad: Steve, for their unwavering support throughout; and for pushing me really hard at the end to get this thesis finished.

Abstract

Analysis of military campaigns through the use of computational models and simulations, is one of the fundamental methods used within defence Operational Analysis at the Defence Science and Technology Laboratory. It helps to develop understanding behind the value of investment so that an agile defence capability can be built and maintained; in order to face the challenges of an ever-changing world. However, many of these models have been used and adapted continuously over decades, resulting in code-bases that have become unmaintainable in the face of constrained budgets. To address this problem, a number of software modelling framework, based upon the reuse of code and concepts have been developed. However, many of these ultimately did not achieve their full potential, because they merely iterated upon the same software patterns which had been used to produce models to-date. The most recent attempt however, known as the Generic Aggregator Model Valuator, was very different in this regard, due to its exploitation of the emerging Object-Functional design paradigm.

As an emerging design paradigm, Object-Functional is still to be more formally understood. There is currently a lack of widely accepted design patterns for the paradigm and explicitly acknowledged examples of its use in projects. This thesis examines the Object-Functional paradigm in greater detail, by performing a qualitative evaluation of defence models built using the paradigm compared to extant models that use other approaches such as Object-Orientation. The evaluation aims to answer two key questions: what benefits does the exploitation of this paradigm bring to defence modelling? But also, what challenges? The Generic Aggregator Model Valuator framework's implementation of the paradigm is also presented in detail, illustrating the patterns it uses, standing as an example that can contribute to the further refinement of this paradigm in the future.

Table of contents

List of tables

List of figures

1 Introduction

1.1 Background

One of the key objectives of defence Operational Analysis (OA) is the study of investment. Should we invest resources into acquiring and maintaining this new piece of equipment? Should we invest manpower into achieving this objective? What are the implications of investing in this course of action over another one? These are just some of the questions that are routinely considered by teams of analysts and modellers working in U.K. defence Science and Technology (S&T). The answers to these questions have the direct potential to inform the thinking of high-level decision makers within the Ministry of Defence (MOD). Therefore, providing both timely and valid advice is a crucial activity within defence science and technology (HM Treasury, McPherson, 2013).

In the U.K. today, one of the main organisations generating advice from the OA process is the Defence, Science and Technology Laboratory (Dstl), of which I as the author am an employee, and who are ultimately the primary customers of this research. Dstl are described as the “U.Ks. leading Government agency in applying Science and Technology to the defence and security of the U.K” (Defence, Science & Technology Laboratory, 2015). Dstl are part of the U.K. MOD and their programme of work enables them to provide advice on a wide range of S&T fields. This includes providing specialist advice on specific low-level military systems, all the way up to high-level OA on defence policy and investment into capability (HM. Treasury, McPherson, 2013), (HM. Treasury, 2015).

Dstl is broken down into a number of divisions, in order to deliver its S&T programme. The Defence and Security Analysis (DSA) division, where this research is focussed, is primarily concerned with High-Level OA (HLOA). Their work looks across all the military domains and activities, in order to provide evidence to support decision makers in MOD, so that they can invest in the most appropriate capabilities and strategies to satisfy current and future defence requirements. As a result, DSA’s work is both cross-cutting and extremely diverse, requiring the application of a wide variety of analytical techniques in order to understand the implications of investing in one approach over another.

However, for all the methods, expertise and resources at its disposal, very often we have found in our experience that it is the amount of available time that is possibly the biggest constraint upon the breadth and depth of OA activities that can be undertaken. OA has to take into consideration, a wide range of military domains[1], fields of research and perspectives when producing its evidence. That evidence is generated using a variety of methods, ranging from soft analytical processes up to more hard approaches, such as complex computer-driven simulations. These specific methods in themselves require the support and expertise from Subject Matters Experts (SME), such as computer science. Defence, as an enterprise, is also extremely large, comprised of many associated bodies and flows of information throughout. All these elements combine to produce a very complex environment to methodologically engage with. It is an environment, particularly in these austere times, that is under great pressure to achieve increasingly more with fewer resources. Therefore, it is important that all those elements that make up the OA process itself evolve, in order to meet these demands.

1.2 Statement of the Problem

One of the main analytical techniques used in the OA process is campaign analysis. As part of this activity, representations of scenarios (military campaigns) are produced within a model (commonly referred to as a campaign model) and analysed using a combination of soft and hard analytical techniques, encompassing a wide range of capability and policy perspectives. These scenarios are specifically designed to provide those conditions that will sufficiently test different areas of investment. These campaign models are often run in a simulation, which are typically very large and complex pieces of software in terms of their code-bases, ranging in upwards of tens of thousands of lines of code. However, current approaches to simulation are ill suited to responding to the evolving defence landscape. As a result, use of these models on analysis studies has become a significant time-sink in the overall analytical process.

[1] These domains include the three primary services that comprise the U.K. armed forces (i.e. the Royal Navy the Army and the Royal Air Force) and all their related military activities. These are typically referred to in short as Maritime, Land and Air within the MOD.

At a very high level, the usage of models within the campaign analysis activity can be viewed as a three-stage process (see Figure 1).

Figure [1] - Campaign Analysis Activities

Modelling refers to the building of a representation of the scenario and the input of data into the model; including key Verification and Validation activities of the representation. This quite often involves adaption of the model in terms of either adding new functionality, or adapting existing ones. Processing is in reference to the actual length of time it takes to generate outputs from the models. This can vary depending on the amount of computing power available to the model; however, historically we have observed that the complexity of the analytical scheme requiring processing often expands to fill the computing resources provided. Nonetheless, to facilitate the processing of more complex analytical scheme, this runtime has been reduced with research investment into High Performance Computing (HPC) and parallel hardware architectures, although the full potential of such approaches is currently not being fully utilised. One of our campaign models, known as the C3 Oriented Model of Air and Naval Domains (COMAND) has been enhanced through lazy parallelism, as outlined by the work undertaking internally by Poulter (2011), such that for each replication of the model, a separate instance of the model is generated and assigned to a processing core. There is currently no parallelism occurring at the individual process level for any of our campaign models, which means that none of the algorithms are being individually optimised. Finally, the analysis stage refers to the time remaining to develop understanding of the problem space from the model, which as stated previously is highly constrained by the prior activities.

The current range of software-based campaign models available within Dstl are typically very large in terms of their code-base (in the order of tens-of-thousands of lines of code) and encompass a vast range of functions in order to build the representations within the scenarios (Moffat, Campbell & Glover, 2004), (Taylor & Lane, 2004). These campaign models are also very old, spanning as far back as 20 years of development history. As a result, much of the coding practice and the understanding contained within the implementation of the functions is reflective of those timeframes, and in many cases has not been enhanced (or cannot easily be enhanced) to embrace new and evolving ideas in the field. This has resulted in a number of significant problems with respect to the continued use of these models, outlined in subsections 1.2.1 to 1.2.7 as follows:

1.2.1 Long setup times

Collectively as analysts we have observed that building a single representation within one of these models takes up a significant proportion of the available analysis time. As outlined within Glover and Toomey (2012), we have estimated internally that this can be as much as 80% of the total project time over the course of a financial year, based upon previous analytical studies. This can be as little as a few weeks for a simple representation, up to many months or more for a large scenario. It should be noted that this single representation is only a test of one Course of Action (CoA), or military plan. Given that the purpose of the model is to test the validity of this plan, any uncovered problems may require both a new plan and a new representation of that plan to be produced. This may require significant modifications to the initial representation, incurring additional setup time.

1.2.2 Long processing times

Due to the complexity of the representations that are being processed, the current suite of campaign models can have a very long run-times, with some campaign model being recorded as taking an entire weekend to produce one set of outputs. Additionally, because the code-bases of these models are known to have implementation problems, they cannot easily scale onto a parallel architecture without significant re-engineering. As previously stated in Poulter (2011), Dstl has achieved some success by running individual model replications in parallel, but this is no more than a brute force effort.

1.2.3 Difficult to adapt

Because the code-bases of these models are based on coding best-practice at the time of their creation, the underlying code is not as modular compared to more modern best-practice and contains a great degree of coupling, which is a common issue for Object-Oriented software as outlined by Ottinger and Langr (2011). As a result, making changes to the model in order to create new ideas, or change pre-existing ones, carries a great deal of risk. In fact, some of our campaign models have now reached a point where small changes would produce significant side-effects across the model code-base, such that they can no longer be safely adapted.

1.2.4 Black Boxes

Due to the presence of coupling, it has become very difficult to track the interactions and dependencies between the components of the model. As a result, there are portions of these models that have become effectively black box functions, a key issue also highlighted by Salt (2008) review of common issues affecting campaign level modelling.

1.2.5 Retention of Knowledge

Because many of these software models are not able to exploit more modern development approaches, and may be dependent on very historic underlying libraries and frameworks, it has become very difficult to retain the knowledge internally in order to recode those aspects of the software that can still be safely changed.

1.2.6 Forced Representations:

Most of these models were validated against understanding and assumptions that held true at the time of their implementation. Compounding this aspect with how difficult it is to adapt these models, the analyst is often forced to use functionality in ways in which it was not originally designed, in order to produce the required representation. To illustrate this with a simple example: if a model does not have the representation of artillery, the analyst may have to simulate an explosion of the correct magnitude by other means. This could involve queuing up an airstrike at that location. However, the analyst may then also have to be mindful about additional behaviours this workaround could induce into the representation. Whilst the airstrike is merely substituting for the effect of artillery bombardment, the aircraft could be

engaged by Ground Based Air Defence (GBAD) along the flightpath. Therefore, some kind of fudge in the data may be required to ensure that this does not happen.

1.2.7 Steep learning curves

The time required in order to learn how to use one of these models can be very long. For our COMAND model, it can take roughly 1 to 2 years to become a competent user of the model, but this is heavily dependent on prior experience of the domain(s) being represented in the model and the complexity of the model in question. This can also a two-part problem:

1. Learning how to set up scenarios within the model in terms of data (the static descriptions of the problem).

2. Learning how the model functions, in terms of its features and what it is capable of representing.

    a. Progressively, this has moved further towards learning how the model does not do something correctly and how to work around those limitations in order to produce a representation that is valid.

1.2.8 Problem Summary

The compounding effect from all of these problems is that the vast amount of resources on an analytical study are currently being devoted to setting up the model, leaving very little time to undertake the necessary analysis. Specifically, within Dstl, it has been estimated that this equates to approximately 80% of the total study time and resources (Glover & Toomey, 2012) based upon study leader experience. These challenges limit the insightfulness of the analytical product that can be generated for the customers.

Everything listed thus far represents the status quo with respect to the issues encountered with our currently available suite of campaign models, which have served us well in the past. However, the types of study that these models were built for typically used to last for a long time, sometimes spanning multiple years. Studies today are now much shorter; often lasting no longer than a year, requiring a much faster turnaround from the analytical process. The landscape of defence is also evolving, with new and emergent capabilities and threats (for example, Cyber) and a

great deal of uncertainty, as outlined within Moffat, Scales, Taylor and Medhurst (2011). Representing these new concepts in the current models is extremely difficult, if not impossible, due to the difficulty in adapting them. Additionally, shrinking budgets are making it progressively more difficult to continue to maintain these models. As a result, the current suite of campaign models is progressively becoming more and more difficult to use.

At present, there is a requirement for a new approach to modelling; one that provides the necessary agility in order to keep pace with the current demands of the defence customer base. The approach needs to move away from the status quo, whereby scenarios are effectively being kludged into the models, towards a more bespoke problems focussed development method (Pidd, 1996), (Sargent, 2005). Efforts have been made to address this in the past internally, as outlined by Robinson and Glover (2006), including the production of the Wargame Infrastructure and Simulation Environment (WISE), and the Defence & Evaluation Research Agency (DERA), Reusable Object Modelling and Simulation (DROMAS). However, these frameworks did not reach their full potential, both in terms of providing the capability and the underpinning computer science to enable them. As will be explored later in this thesis, these frameworks iterated upon previous understanding of software development, rather than overhaul it. This means that they still experienced issues such as the coupling between components, which for the modellers and analysts within Dstl, is the key issue to be managed in order to achieve the desired adaptability. Any solution going forward needed to examine the cutting edge in terms of software development, in order to improve upon many of the software limitations listed above and to future proof the solution.

As a result, Dstl's efforts began to move away from constructing models from the ground up as it had been doing up to this point and move towards a solution that was based upon the reusability of ideas (Ansell & Glover, 2008). This would take the form of another framework, with a plug-and-play style approach to components, the benefits of which have been examined in detail in works such as Fletcher, Lukman and Hodson (2005). It was believed that using a framework approach would enable models to be rapidly constructed for the purposes of the study in question, as opposed to the status quo whereby representations are being forced into the models.

1.3 What would solving this problem enable?

1.3.1 Improvements to existing methods

By using a framework approach based on the ideas of plug-and-play, from which all models will be constructed, it was hoped that the following improvements would be possible:

- Higher Modularity – By making the components of the framework much more modular it would become easier to add new concepts to a model and adapt pre-existing ones. The impact of the addition of functionality and changes to pre-existing functionality will also be easier to verify and validate, provided there is sufficient transparency of the interfacing between components.

- Model Reuse – Rather than constructing an entirely new model each time a study makes a request for one; the ideal solution would be to quickly adapt a pre-existing model for a new purpose, so long as the requirements of the study are similar enough.

- Improved Maintenance – If all the campaign models are being derived from the same framework with similar conceptual underpinnings and all of the components are sufficiently modular, it would become easier to maintain the full spectrum of models. All models would be written in the same programming language and share a common architecture, which would ease training and retention of the understanding behind how the models work. The current range of campaign models are all fundamentally different in this regard, meaning that the developer for one model cannot easily transition to supporting another model without a significant reading in period.

1.3.2 New Opportunities.

In addition to improving the status quo, it was believed that a plug-and-play framework approach would enable Dstl to further develop its capability and achieve more from the analytical process, including:

- Providing more time for the analysis – By providing agility to the implementation of models, it was hoped that the available time to conduct the important analysis would grow. Currently the available time only allows us to examine a

sub-set of cases, with a lack of confidence that this investment is being targeted towards the correct areas of interest.

- Sharing of capability – It was believed that a framework approach could evolve into an environment whereby modelling components can be shared internally and externally to the organisation. Mechanisms to enable this sharing are actively being developed, such as the High-Level Architecture (HLA), as outlined by Dahmann (1997); however, without a consistent modelling environment that promotes reuse and interoperability between capabilities, these cannot be easily exploited (Fujimoto, n.d.).

1.4 Enter GAMOV

The resultant framework that was produced to address the issues listed previously was the Generic Aggregator Model Valuator (GAMOV) (Ansell & Glover, 2008), (Glover & Toomey, 2012). GAMOV was built to be a modelling framework, containing within it a set of reusable components, with an associated Application Programming Interface (API) to enable users to build models from those components. In that sense, the capability of GAMOV is comparable to a programming framework, such as the Microsoft .NET framework, albeit with a very defined context and purpose in mind. The details of GAMOV will be discussed later in chapter 4, and models produced from the framework shall be evaluated as part of a case study of this thesis.

The key enabler that made GAMOV more adaptable when compared to previous attempts at building a framework was the approach it adopted for its software architecture, which was an emerging software design paradigm known as Object-Functional.

The goal behind GAMOV was to eliminate the coupling experienced in both our previous models and attempts at developing a framework. As presented in Glover and Toomey (2012), our proposed solution to this was to separate data from functions as opposed to encapsulating data and functionality in the same object. We realised this by having two different categories of object, those that contain data and those that contain functions, which are characteristic of the Object-Functional approach as described by Sousa and Ferreira (2012). The key idea was that functionality within the model would become more service-driven and thus reusable around the model by multiple object types.

1.5 Aims of this research

The principles behind the Object-Functional paradigm as outlined by Sousa and Ferreira (2012) is to blend the Object-Oriented and Functional paradigms together in order produce new families of patterns for organising code, which they hypothesised to be less prone to the issues that can arise from misusing Object-Orientation patterns. This thesis shall explore the Object-Functional paradigm in greater detail in order to illustrate how its exploitation enabled GAMOV to succeed in its aims, compared to previous models using Object-Orientation alone.

Within this thesis the current state of the art with respect to defence modelling approaches shall be examined in order to illustrate key characteristics of their implementation. The state of the art surrounding both the conceptual understanding of Object-Functional and its implementation shall be examined in order further illustrate the gaps that this research can contribute to.

The concept of GAMOV and how it implements Object-Functional shall also be presented, identifying the key patterns used in its structures and how these allow it to conform to the aims of the wider paradigm. This case-study shall serve as example to contribute to the very much evolving body of knowledge surrounding this paradigm and what combinations of software patterns could be used in conjunction with it.

Finally, this research shall evaluate some models produced using the Object-Functional approach via GAMOV, compared to extant models in Dstl that adopted more historic approaches such as Object-Orientation. Each model shall be presented as a case-study and analysed in order to understand how their implementation choices affect their capability and where Object-Functional has or could contribute further.

The purpose of this evaluation shall be to answer two key research questions:

- What benefits does the Object-Functional paradigm bring to the development of defence models?
- What challenges does the Object-Functional paradigm pose for the development of defence models?

2 Literature Review

2.1 Chapter Introduction

In the introduction to this thesis, the concept of defence modelling and the purpose it serves in the wider context of delivering analytical products to defence customers was overviewed. Some of the classical problems empirically observed at Dstl with our extant bespoke capabilities were also presented and how this has led to the desire to move towards a framework approach to building our future models. This section shall now examine key contributions from the literature with respect to the development of defence model frameworks, in order to establish what the key characteristics of these are, how their implementation impacts what they deliver and how GAMOV under the Object-Functional approach may add new insight to this body of knowledge.

Following this the field of Object-Functional shall also be examined in further detail; specifically, how is it currently characterised? What work has been undertaken to understand more about it? And what the implementation of such an approach looks like?

2.2 Defence Modelling Framework Development

In order to improve the agility of the analytical process and to increase the reusability of both the implementation of models and their conceptual underpinnings, both the MOD and other defence bodies around the world have been developing solutions in the form of software modelling frameworks. This section shall review some key examples of these. For defence, these can range from single defence domains and research areas, all the way up to campaign level, which pulls together many of these single domains into one representation. The latter tends to be much more challenging, because many different domain specifications are being pulled together within a single representation. This is confirmed by Teo and Szabo (2008, p.103) who stated that “Component-oriented frameworks exist for particular application domains but cross-domain component integration or semantic composability remains an open issue”. To illustrate an example that would be seen in a campaign level model, it may be common to blend land-based combat modelling supported by air assets, which may include some air-to-air based engagements. Quite often in the models at Dstl, the conceptual frameworks for these domains differ enough to make composition difficult. For example, the land-based combat may be easily abstracted to occur on a node and arc environment; however, the air assets could be operating in a positional

based, longitude/latitude system. This may then require the incorporation of a command and control system for each of the two domains, which is planning on the assumptions of two different environments. This makes analysis of these models very difficult as outlined by works such Hoffman, Palli and Mihelcic (2011) through the use of applied semantics. Because the conceptual foundations of each of these domains have different specifications, the software foundations are forced into operating at cross-purposes.

Comparatively, frameworks for single domains tend to be much more successful in producing reusability of their components because they have similar conceptual foundations. As described by King, Hodson and Peterson (2017, p.4156) “If software is written to implement a specific conceptual model, there is usually good alignment between the model being created and the envisioned one”. For example, the Chemical, Biological and Radiological (CBR), Virtual Battlespace (CBR-VB) presented by Lloyd, Newton and Perkins (2014) from Dstl, showcases a successful reuse case of components and models within a single domain, through the use of a synthetic environment. Their work demonstrated that reusing models within the CBR domain was easily achievable due to the similarities in requirement and conceptual underpinnings. However, they did encounter issues when trying to cooperate with models from the domains of acquisition and advice because of the subtle differences in their conceptual underpinnings. Whilst they were able to overcome these issues through code refactoring, this highlights an example of the implementation issue that can arise when different domains interoperate with one another. Frameworks such as this have also seen success in collaboration and cooperation with other institutes via the HLA, as described by Dahmann (1997). The HLA is a “common architecture for reuse and interoperation of simulations” allowing for components in a model or simulation to share their data and insights. As per Lloyd, Newton and Perkins (2014), frameworks such as CBR-VB can plug into the HLA and cooperate with other CBR based models and simulations, because of the similarities in the conceptual underpinnings. However, the campaign models representing multitudes of different domains that are produced in Dstl DSA division, struggle to exploit initiatives such as the HLA. Depending on the configuration or framework by which these domains are organised within the software can produce descriptions that are different enough so as not to be compatible with those in other organisations. However, compounding this issue further is the overall architecture and code organisations of these models (as

overviewed in chapter 1), making it difficult or near impossible to modify the conceptual underpinnings to become more compatible. This is where a framework such as GAMOV, which will be explained in detail in chapter 4 is attempting to address this. King, Hodson and Peterson (2017) highlight that in many of the frameworks being developed, not enough consideration is being made to the overall software architecture of these solutions. GAMOV is an attempt to go to the software level to leverage the adaptability of both its components and the framework for their organisation. Exploitation of Object-Functional (as will be examined later), is believed to provide a loose but explicit coupling between components, which would allow all functional descriptions of the resultant model to be changed. As a result, it is hoped that GAMOV models will have a better chance of cross-domain cooperation and exchange across an architecture such as the HLA, because the conceptual underpinnings can be leveraged at the algorithmic level and changed. Whereas for many of our extant campaign models, the conceptual framework is fixed, because the coupling between model components is preventing significant changes.

At Dstl there have been efforts made by predecessors to the DSA division to develop modelling frameworks for building campaign level models covering multiple domains. Examples of these come in the form of WISE and DROMAS (Robinson & Glover, 2006). However, the full potential of these solutions for the development of campaign level models has not been achieved due to the lack of advances made in computer science at the time (Glover & Toomey, 2012). Within DROMAS for example, considerable effort has been made to provide a framework that produces models with a consistent look-and-feel as well as a comprehensive suite of functionality for specific problem types (Nesfield, 1998), which in the case of DROMAS is the representation of peace-support operations. However, this considerably constrains the range of solutions that can be developed using the framework components. If the idea of a reusable component framework were analogous to Lego blocks, DROMAS would be an example of a very specific Lego set, such as car kit. Whilst it may be possible to change the position of blocks that represent the lights, the direction of spoiler and the position of the doors, the resultant product is always going to unmistakably represent a car. DROMAS is an example where the architecture has been focussed on addressing the non-functional requirements of the resultant model and not enough focus has been given to understanding the computer science that enables the underlying interaction of the components. As a result, problems at that

level may have permeated through the rest of the capability and cannot be easily addressed within practical budgets.

Externally to the U.K MOD, organisations such as the U.S Department of Defence have developed approaches such as the Advanced Framework for Simulation and Modelling (AFSIM) as outlined by Clive, Johnson, Moss et al. (2015). AFSIM is a component-oriented framework that provides reusable model services that are common across all models, both in terms of construction of the model's foundational services (e.g. time management, random number generators etc.) and the entities that operate within the resultant simulation. The idea here appears to be that a modeller can reuse many off the shelf components built around well understood concepts in order to construct a model. Entities within AFSIM appear to be broken down into component hierarchies, such as the Entity itself (e.g. tank, plane) and the combination of sensors, weapons and platforms that it is made up of (Clive et al. 2015, p.74). However, whilst the components in a framework such as AFSIM from their descriptions appear to be malleable through data and the model framework can be configured by their organisation, there is no indication as to whether the "functional architecture" (Clive et al. 2015, p.74) allows for the leverage of the algorithms for replacement or adaptation. In Dstl this has often made the exploitation of component frameworks difficult for the campaign level problems. As a purely speculative example, with AFSIM being a U.S endeavour, if it were to have U.S military doctrinal assumptions built into the components, it would make it difficult for representing U.K approaches unless these can be changed. This is where exploitation of a GAMOV approach becomes desirable from Dstl's perspective, because these concepts can be changed if they are fundamentally different and can be further changed as defence policy and doctrine evolves over time. However, this would come at a cost of training overhead for different communities of user. By providing an approach that is customisable at the algorithmic level, the modeller, who will likely be an analyst by trade, may be required to write more code; whereas in a framework like AFSIM, this appears to be more abstracted from them. Using a GAMOV type approach based on Object-Functional thus may mean analytical studies need more diversification in their skill-sets to leverage its full potential.

Another example of a component style framework developed for defence is the Simkit framework, outlined by Buss (2002). This work makes a similar claim as with Object-Functional, that the resultant framework has high reuse potential and a loosely

coupled architecture. Simkit achieves this through the use of what is described by Buss (2002, p.243) as “Listener Event Graph Objects” or the “LEGO component framework”. The simple description of the methodology behind this is that event listeners will trigger certain sequences of events or functions when the preconditions defined by the listener have been met. Compared to GAMOV (described in chapter 4), this appears to be a more event triggered approach, whereas GAMOV is more scripted in this regard. GAMOV schedules a check to be performed for certain events on a user defined interval. For example, it will check for instances where a location has two or more entities of opposing sides within it on a time-step and then trigger the defined combat algorithm for the simulation. The descriptions of how Simkit works are framed around nomenclature for discrete event simulations, where there is state and a state transition, but the notion of what they are trying to achieve is similar to the characteristics of Object-Functional (as will be described in section 2.3) where there are components with state and events (functions) are triggered to perform a state transition. There is evidence here to suggest that there may be some implementation patterns in the code used in Simkit that could also contribute to Object-Functional that are not explicitly described in its literature and documentation. However, the approach to triggering and scheduling calls to functions or methods is different to GAMOV, which could serve a useful future comparator to the GAMOV pattern for triggering events in an Object-Functional environment.

Finally, an example of a component framework that is attempting to address the multi domain problem is the Composable Discrete-Event scalable Simulation (CODES) presented by Teo and Szabo (2008). CODES is focussed on delivering composability of its components through the use of semantics Teo and Szabo (2008) cite Petty and Weisel (2003) two categories of model composition as being either “syntactically” or “semantically” composable. CODES is focussed on the latter by providing an ontology and a semantic language to allow components to be described, and for other components that can exchange under the same semantic structures to be discovered. This is different to GAMOV, which under this definition would be a syntactical based approach, where the underlying structures are interfaced with code for composability rather than semantics. A semantic based approach for GAMOV could be future stepping stone, however, none of our extant models have any form of semantic encodings. It would also be difficult to apply semantics to these given the

difficulties in breaking down the underlying code structures, which could limit the discoverability side of things.

2.2.1 Summary

Upon review of the frameworks examined in this section, there is clear intent by all the solutions to promote reuse, but the approaches vary in the implementation. Key characteristics from this survey that make up and differentiate these solutions, including GAMOV appear to be the how the concepts of entities (actors), components (the building blocks) and frameworks (the protocols) are applied in the various solutions.

DROMAS (Robinson & Glover, 2006), (Nesfield, 1998) for example is very much oriented by its framework, defining very rigidly how its components are composited in order to produce the resultant model. As a result, there is very little flexibility in how the resultant actors behave and can be adapted.

AFSIM (Clive et al., 2015) focuses more on components where new entities can be built from off-the-shelf components. However, the functionality of the entities still very much appears to encapsulated. However, what appears to separate these from GAMOV at least in terms of their concepts are the underpinning frameworks of components versus entities. Taking AFSIM for example (Clive et al., 2015) frames its simulation around components, and what would be the actors in the simulation (or entities) are made up of some composition of these components drawn from object hierarchies. Therefore, a component to form a vehicle for example is some kind of fusion (or referential linking) to other components that make up the capabilities for that vehicle, in the form of sensors, weapons etc. Additionally, a component may be the fusion of one or more algorithms in order to realise an overall capability. GAMOV (as will be examined in chapter 4) takes a subtly different approach, because the algorithms are the highest form of component in its entity framework. Thus, the entities are a fusion of attributes plus various algorithms they can exploit. This then defines what the entity is in the context of the simulation, by what is it able to do, rather than what it is called. An analogy to this would be the concept of duck-typing found in interpreted language, where objects are classified by the language's typing system by how they behave rather than what they are.

The CODES framework presents an example of the distinction that can be made between frameworks being syntactical versus semantic composable. CODES is focussed on the angle of composition through components that can natively exchange data through shared semantics, which is how it is targeting its reuse case. By this definition, GAMOV as a defence modelling framework would be seeking to achieve the syntactical approach to describing its components in order

Finally, the Simkit framework presented by Buss (2002), demonstrated a very similar solution that is characteristic of what GAMOV is trying to achieve under Object-Functional; though Simkit is described in terms of discrete event simulation nomenclature. Presenting GAMOV as an explicit example of the Object-Functional paradigm and its organisation could contribute to other frameworks with similar approaches being able to acknowledge their underpinning software organisation in similar terms, which could enable more patterns to be contributed alongside GAMOV to further shape the paradigm.

2.3 Object-Functional

2.3.1 What is Object-Functional Conceptually?

At the conceptual level, the Object-Functional paradigm as currently understood is a hybrid paradigm combining the Object-Oriented and Functional programming paradigms (Sousa & Ferreria, 2012). For the longest time both of these paradigms have evolved in relative isolation, due the difference in the goals behind each (Odersky, 2014), but in practical terms there is little reason for them to be mutually exclusive. The benefits and limitations of each are often compared against one another (Harrison, Samaraweera, Dobie & Lewis, 1996) in terms of code quality and performance; and sometimes the two have been combined through layering, not necessarily blending (Kristensen, Hansen & Rischel, 2001). Object-Functional attempts to unify the two paradigms in order to yield the benefits of both whilst eliminating their shortcomings (Lau, 2015). In other words, it is seeking to harness the structure and descriptive power of Object-Orientation and the stateless characteristics of Functional. The perceived benefits of combining these two paradigms, as hypothesised by (Sousa & Ferreira, 2012), is that Object-Functional would produce leaner and better-quality code, and more importantly the elimination of what is referred to as anti-patterns. This is the key characteristic for tackling the problems with software maintenance in a domain like defence modelling head-on.

Anti-patterns as described by Abbes, Khomh and Gueheneuc (2011, p.181), are "poor solutions to recurring design problems; they stem from experienced software developers' expertise and describe common pitfalls in object-oriented programming". They also state that "Anti-patterns are generally introduced in systems by developers not having sufficient knowledge and-or experience in solving a particular problem or having misapplied some design patterns". Many of the classic anti-patterns that can occur within object-orientation specifically are listed in the works of Webster (1995). One of these patterns, described as "spaghetti", is where dependencies exist between the components of the software, which is characteristically similar to how models in Dstl have coupling between their components. Code with the spaghetti pattern makes traversing the code base more complicated due the lack of interfacing between the coupled elements, which can make adaptation difficult, because a change in one component may require changes in another, and overall makes the process of finding the root cause of bugs more time consuming. This is an aspect empirically observed by modellers and developers in Dstl with respect to our campaign models. Other anti-patterns include "Blob", as described by Abbes, Khomh and Gueheneuc (2011, p.181), as a "large and complex class that centralises the behaviour of a portion of a system". Abbes, Khomh and Gueheneuc's (2011) research into anti-patterns examines how these patterns affect software maintenance, by running experiments on software containing single or multiple anti-patterns. They concluded that software with one anti-pattern such as spaghetti only compound software maintenance but it's when there are multiple patterns present that actual breakages begin to occur; for example, spaghetti mixed with a blob pattern. However, their research is purely framed around object-orientation being the source of anti-patterns. If the source of anti-patterns is through mis-applying design patterns, then the Functional paradigm and indeed the Object-Functional paradigms could also have their patterns applied incorrectly to produce unintentional behaviours in the software. However, before the Object-Functional paradigm and its patterns can be misapplied, they need to be understood and defined, which as highlighted by Sousa and Ferreria (2012) are limited at best and still evolving.

The fundamental idea behind Object-Functional at present seems to be about splitting entities within programs that hold state, from those entities that transform state (Sousa & Ferreria, 2012), (Lau, 2015). In Object-Orientation terms this would mean breaking (but not discarding) the classical approach to encapsulation down into

two object categories. The first category of object is used to represent the data elements of the program (i.e. the attributes that hold state). The second object category is used to hold the transformative functions of the program, which is where at the fundamental level the Functional paradigm meets the Object-Oriented paradigm.

To illustrate this, consider a very simple Object-Oriented example expressed in Uniform Modelling Language (UML) with a defence centric entity, such as a tank. In Object-Orientated design, an approach to implementing a tank would be to declare a class in some way that may take the following form:

Figure [2] – Tank Class under Object-Orientation.

Object-Orientated languages are built upon the fundamental idea of using a construct known as an Object. The roots of this concept can be traced back to the development of the Simula language (Dhal, 2002). In a true Object-Orientated language, absolutely everything that is created, manipulated and destroyed within a computer program is an object. Object-Orientation is a reasonably well-understood paradigm, with many design patterns in existence to communicate its concepts to other software developers (Sousa & Ferreira, 2012). Following this notion, the resultant tank object has attributes that describes what its relative speed is, what its firing capabilities are, how much armour it has etc. It also has methods that could be used to transform these attributes. For example, as the tank takes damage, its overall combat effectiveness will decrease in some way, therefore the method would modify this attribute to the new value. The derivation of this object may have occurred as a single class purely through encapsulation, or it may be a subclass of another grouping, such as vehicle, and inherited its attributes and methods. Either way, the resultant class as far as the describing the concepts here are sufficient as illustrated in the above

diagram. The class also defines the access control model for the object, indicating whether attributes or functions are private to the object itself, or publicly accessible by other objects at various levels.

Whilst encapsulation is a powerful and easy concept to grasp in terms of implementation within object-orientation, it requires a certain degree of discipline and good design skills from the programmer to be effective (Kester, 1993). In particular, careful consideration of the types of objects that the programmer is creating and how the attributes and methods are reused around the overall system. Failure to do so is what can lead towards the emergence of anti-patterns in the code. Taking the example of the tank, it has methods that enable it to move and to fire. However, at a conceptual level, the movement of the vehicle and firing of a weapon are not that complicated in terms of the overall effect they produce, particularly in a campaign model, which tend to be highly aggregated representations that do not include the physics of how these systems work. At this level of aggregation, the movement of a tank is no different to how any other entity could move through their environment. The resultant speed, distance moved etc. are merely a product of the terrain they are moving though. The developer could solve this problem a number of different ways. They could declare these methods as part of a superclass such as ‘vehicle’, which would allow tanks, planes, boats etc. to all use the same generic methods through inheritance. However, there are other sub-optimal ways, such as having one object class such as the tank becoming the owner of the movement and fire methods and other objects simply calling upon these methods. Not all issues with Object-Orientation may be down to misunderstandings of the pattern, but down to external pressures. A developer may have to compromise some technical debt within the implementation of their objects in order to meet a more immediate deadline however the reasons for choosing to accept this debt and not repair it upon occurrence is very much dependant on the methodological environment (Soares de Jesus & Vieira de Melo, 2017). Whilst this very much a basic example, when coupled with external pressures such as time and cost, quick wins of this nature can occur and have been empirically observed in models at Dstl. This is what produces the coupling described by Ottinger and Langr (2011) or the spaghetti code anti-pattern (Webster, 1995).

Staying again with the example of the tank, the concepts of Object-Orientation have been communicated using real world concepts, which is a common approach found within a lot of teaching material. In this author’s experience with respect to modelling

within Dstl, the potential problem this can lead to is that it can influence the thinking of the developer in unhelpful ways. In other words, a developer may develop practices whereby they use objects only to describe real world concepts, forgetting the fundamental notion that everything within an Object-Oriented language is an object. This can lead to situation where programs have explicit descriptors for the nouns (what the object represents) but not always the verbs (what the object does or the actions that influence the object i.e. the methods). The concept of nouns and verbs is a common analogy within Object-Orientation (Drobi, 2007), but not always the best and most comprehensive. Also Object-Oriented design is primarily focussed around the notion of nouns. Object descriptions via classes are nouns, and only the methods are verbs, which are owned by the nouns (Yegge, 2006). This is why some have questioned whether Object-Orientation by itself is sufficient in order to fully describe the world (Mansfield, 2005) or even failed in its goals entirely. Verbs should be their own objects, not a dependency of some larger construct and should be able to be passed as arguments around a program in the same way as their noun counterparts. This has resulted in realisation of the 'First-class object' construct appearing within modern Object-Oriented languages (Van Rossum, 2009). This particular limitation of the object-oriented paradigm has been raised in works as far back as Jalote (1989) highlighting a need for more flexibility in the object-oriented paradigm through a set of extensions in order to accommodate these complex real-world concepts. Jalote (1989) proposed two extensions to the object-oriented paradigm. Firstly, the process of "functional refinement" which includes identifying and breaking these transactional (highly generic methods) away from data sources; in other words, breaking the encapsulation. This would allow for methods to become reusable functions as per the Functional paradigm. The second was identifying the need for objects to be nested in some way, because the real world is also formed of complex hierarchies of subordinates, as opposed to the model of inheritance. Whilst Jalote (1989) does not explicitly refer to a new paradigm, opting to characterise these as extensions to the Object-Oriented paradigm, the objective of "functional refinement" is strikingly similar to the fundamental idea of Object-Functional. However, this notion of functional refinement is more focussed on the process of identifying those functions that could be broken down more generically, with still a large emphasis on complex objects hierarchies holding and sharing functionality. Works such as Qian, Fernandez and Wu (1995), who attempted to build upon the work of Jolote (1989) found this to be the case when they applied these processes to a component framework for the medical

domain. Additionally, Jalote (1989)'s highlighting for the need for nested objects also falls in-line with both defence hierarchies and how these are implemented in GAMOV, which shall be examined in chapter 4.

Therefore, if we take these concepts of Object-Functional as described (Sousa & Ferreira, 2012), (Lau, 2015) and apply them to the example of the tank. The resultant object organisation would look like the following UML diagram:

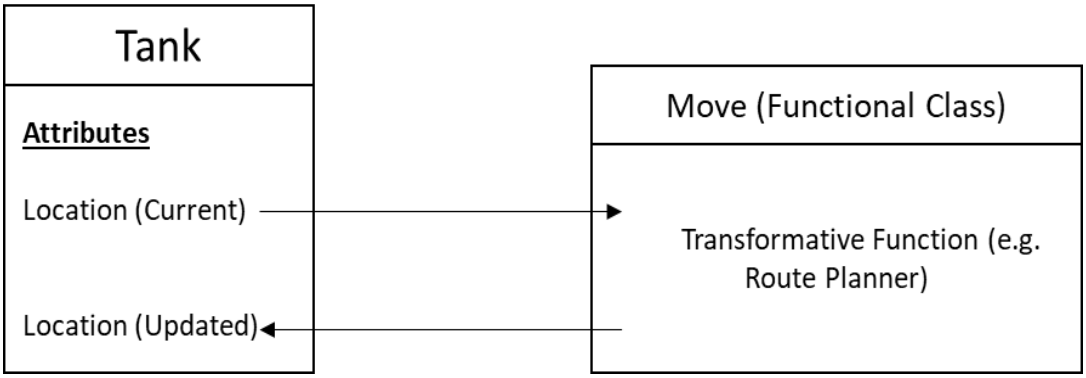

Figure [3] – Tank Class under Object-Functional.

In the Object-Functional approach to object creation, the 'tank' object is now merely an owner of state and does not have any methods. Each of the methods that would normally be encapsulated within pure object-orientation are now broken out into their own object's classes. This leads to a much more decoupled code organisation, because now these methods can be exploited by anything that needs to move, or fire a weapon etc. These methods, as they would be called under object-orientation are now functions under the nomenclature of the Functional paradigm. In other words, discrete services that merely changes the values of data holders.

Through this organisation of the program elements, the state becomes de-coupled from the functions that are used to transform them. This appears to remove the majority of the common problems of coupling in Object-Orientation at the point at which the program elements are first organised. In other words, it helps to eliminate or reduce common anti-patterns hypothesised by Sousa and Ferreira (2012). Whilst the elements of the program and their interactions are still coupled, it is explicit and loose, which was identified in works such as Ottinger and Langr (2011) as being the key to removing the tight coupling found in traditional Object-Orientation. This means that elements of the program can be changed limiting the side effects on the rest of

the system (or the generation of said side effects and their spread becomes explicit). It also makes the nouns and verbs of the program equal in terms of how they are used, albeit the implementation mechanism within the languages is slightly different. Nouns are still classes with respect to traditional Object-Orientation, whilst verbs are a special case of class with respect to Functional.

2.3.2 Object-Functional Implementation

The implementation approaches available to achieving Object-Functional design are not necessarily new ideas, as outlined by Kontio, Mäyrä and RÖnkkÖ (2007); and have progressively been included in programming languages for many years in the form of “functional classes”. This is where an object as per the object-oriented paradigm is used to house an independent function commonly found in the functional paradigm. This resultant functional class can then be called in order to “mediate” Kontio, Mäyrä and RÖnkkÖ (2007, p.) a data transformation. This could be a change to one entity or a change to multiple entities that are interacting with one another. The notion of a functional class also exists in other languages, such as Python, under different nomenclature through the use of the first-class object pattern. There is also another informal description for this construct referred to as the “Functor object design pattern” (or function objects) (Van-Rossum, 2009), which originally stems from the Functional paradigm. Regardless of the nomenclature used: functional classes, first class objects and functor objects are a single but key design pattern recognised as being part of the Object-Functional paradigm. However, there is very little more beyond this in the paradigm as currently described. The Object-Functional approach as described in literature like Sousa & Ferreira (2012) provides merely a conceptual notion of data and functionality being split into their own objects using functional classes, but not any further detail about the implementation of these functional classes. For example, Kontio, Mäyrä and RÖnkkÖ (2007), describe other patterns such as the “memento” pattern as being fundamental to any mediation by a functional class. This is where a copy of the object being transformed is made before its attributes are changed, enabling a rollback process if required. However, the work presented by Kontio, Mäyrä and RÖnkkÖ (2007) is focussed around databases, where storing the prior state would be important if you are making changes to underlying data that lots of services are reliant upon; therefore, the context of applied patterns is also important. The usage of these patterns is not currently explicit in the

wider context of Object-Functional. The absence of these patterns and the permissible combinations are a key gap

2.3.3 Object Functional and Service Oriented Architectures.

By accepting the notion of clearly separating the elements of your program into objects that are state and objects that are transformative functions through Object-Functional; an interesting comparison can be made to some of characteristics of the Representation State Transfer (REST) network architectural model presented by Fielding (2000).

The concept of REST is focussed around the separation of clients and services over a distributed architecture, whereas Object-Functional is for the most part concerned with the separation of code found within the same application namespace. As a result, there are some fundamental differences in terms of scale and REST has unique considerations associated with network architectures to contend with. However, upon a more detailed analysis of the two paradigms, their core characteristics have strong similarities. By considering the following characteristics of REST in Fielding (2000), this is how Object-Functional compares:

- Client-Sever - Whilst this is meant to be framed against the idea of a network of client machines communicating with a server; in Object-Functional, the data objects become the clients and the functional classes become the servers.

- Stateless - Due to the stateless nature of the core internet protocols, REST frames the design of services to be agnostic to the clients they are providing their services to. Additionally, the communication between clients and servers is fundamentally stateless; in other words, the servers have no idea what they are speaking to, they are merely providing their services to a data request. Within Object-Functional this is the same. Objects containing state make a request to functional classes, who then process and return the result to the state object. These functional classes are unaware as to what they are speaking to conceptually, they are merely servicing a request.

- Cache - Within REST, this specific criterion pertains to the ability for clients to cache the response from the server for further use. This is to accommodate for the fact that as Fielding (2000) describes that network efficiency would be

degraded if the client was retrieving the same information multiple times from the server. For Object-Functional, on the surface this seems like a non-applicable criterion, however, there are modelling concepts where caching the result within the state objects would be useful. For example, route finding algorithms used to derive options for moving around the environment often produce matrices of possible routes. If this output were cached with the entity that is moving, the functional class responsible for movement could consult this rather than having to call a route finder function every time a new decision point along the route is reached.

- Uniform Interface - In order for remote services to be reusable, a common interface is needed to ensure that the output of services can be used by all clients. This is also be important for Object-Functional systems, because the input and output interfaces for functional classes need to be consistent in order to promote their reuse.

- Layered System - Fundamentally this is the notion of laying services to control the side-effects produced by different groups of services. This is not explicit within the Object-Functional paradigm, but would be a very important conceptual control mechanism. For example, there may be functional classes that are not permissible to be used on certain problem specifications and thus should be layered away. The importance of this will be drawn out later in the evaluation of Object Functional models produced using GAMOV in chapter 6.

- Code on demand - This characteristic of REST is the least applicable, because the original notion of this requirement was focussed around code in the form of applets, which is now less common in a post HTML5 and JavaScript world. However, the notion of passing functionality to a client to extend its capability may be a spin on the Object-Functional notion of client-server separation. So far, the literature frames the paradigm on services being remotely called, but there may be an implementation pattern where copies of functions are passed to clients. This would also be described by Fielding (2000) as an “oxymoron” approach, but the impact of the pattern is an avenue that could be explored.

In fact, much of my earliest exposure to Object-Functional originally came through applying the notion of REST to Object-Orientation with respect to the Dstl models

(Boakes & Toomey, 2012) and ultimately GAMOV before the realisation that Object-Functional exists. Whilst there is no explicit link to indicate that the two paradigms have influenced one another, the comparison between REST and Object-Functional at least indicates that both the world of web technologies and the more traditional approaches to programming are becoming more unified in terms of component organisation.

2.3.4 Object-Functional Languages

In terms of languages to express the Object-Functional paradigm, these are also still very much evolving. The most prominent language in support of Object-Functional programming is the Scala language, pioneered by Odersky et al. (2006) at the Swiss Federal Institute of Technology in Lausanne (EPFL). Scala is described as a hybrid language seeking to unify the capabilities found within functional languages with those in Object-Oriented. Whilst Scala is the prime example of an Object-Functional language and is gaining significant prevalence within the industry; other smaller language projects are also being undertaken, such as the case of SARD, which seeks to implement non-Java virtual machine implementations of a Scala style language (Svallfors, 2011).

The Scala language, as per its name: Scalable language also highlights another area of benefit brought about by Object-Functional, which is its scalability onto multicore architectures. The capability of this was demonstrated by Pankratius, Schmidt and Garreton (2012) showing that the capabilities of Scala combined with the Object-Functional nature of the programs made writing code for these architectures easier when compared to more strict Object-Oriented languages like Java. This is due to the fact that first-class objects can be assigned to processes and thus hardware resources trivially. This is also comparative to the ideas of tuple space languages, such as Linda, where the organisation of code is similar to Object-Functional in terms of separating those things that are data, from those things that use data, to enable effective message passing (Ahuja, Gelernter & Carriero, 1986). Scala applies these ideas with the use of frameworks such as Akka (Lightbend Inc, 2011), which enables effective message passing across components arranged in the Object-Functional organisation to achieve high scalability.

However, as highlighted by Kontio, Mäyrä and Rönkkö (2007), many of the mechanisms for implementing functional classes have existed for quite some time in other languages. Python, Ruby, C# to name a few have the concept of the first-class object, or the functor pattern, which enables them to package a function inside an object, thus enabling the basic implementation of the Object-Functional paradigm. However, because these languages are primarily Object-Oriented in nature, the range of functional capabilities on offer may seem deficient when compared to a language like Scala. However, the basic notion of how much Object-Oriented capability versus Functional capability should be used in an Object-Functional solution is not explicit, due to the lack of information surrounding pattern usage within this paradigm. The goal of Object-Functional currently is to unite the paradigms, but how much of their constituent functionality is required will ultimately depend upon what the developer is trying to achieve with it.

2.3.5 Why use Object-Functional for defence modelling?

Principally, Object-Functional ended up being the choice for the GAMOV approach due to one fundamental criterion, which was the desire to eliminate the coupling (or anti-patterns) inherent with code-bases of our models. As stated previously, REST was the original approach that was applied to GAMOV because the notion of client-server separation and stateless communication seemed to deliver what was required in this regard. It was not until the Object-Functional paradigm was identified and associated with REST that the importance of other characteristics of the REST protocol began to become prominent. Sticking with the notions of anti-patterns it may have been possible to realise a similar solution with just a purely Object-Oriented approach provided careful application of its patterns are applied; however, Object-Functional seems to eliminate many of the common anti-patterns at first principles.

Due to the lack of explicitness in the patterns that make up Object-Functional and the difference in nomenclature being used, it is possible that other frameworks in defence are exploiting the paradigm but not explicitly acknowledging the fact. For example, Clive et al. (2015), describe that the AFSIM framework is built with “modern programming paradigms in mind”, but the majority of their work is framed solely in both the conceptual ideas and nomenclature of Object-Orientation. However, the implementation mechanism by which they have enabled their functional reuse and the patterns they have used is unclear. As the framework is pitched very much

around providing components, rather than algorithms, it could be that these functions are just reusable objects containing both attributes and methods as per Object-Orientation; or there is a lot more generality in their make-up conforming more to the Object-Functional paradigm. The former is more probable due to the hierarchical nature of the component descriptions. However as per Sousa and Ferreira's (2012) observation that there is a significant lack of explicit acknowledgement of Object-Functional usage, frameworks such as AFSIM may not have a baseline to compare their implementation against. The application of Object-Functional to a framework such as GAMOV could serve as a key contribution in this case in order to compare implementations and potentially refine those in other frameworks. This would also apply to the example of Simkit (Buss, 2002), which as stated earlier is characteristically similar to Object-Functional but uses different nomenclature and framing.

2.3.6 Object-Functional Summary

Overall the literature relating to Object-Functional is very much in a state of evolution, due to the continuing emergence of this paradigm. It is clear that the aspirations and some of the key characteristics of what comprises an Object-Functional program are beginning to be understood, but as highlighted by the thesis proposal from Sousa and Ferreira (2012), there is a lack of software design patterns in order to communicate specific implementations of the paradigm to other developers. It is believed that Object-Functional provides a solution to removing the coupling experienced within Dstl models and thus is why it was chosen as the basis for GAMOV.

The organisation of the resultant code as highlighted previously separates transformative functions away from the state they are manipulating. This results in a code-base where the linkages are explicit but the coupling is extremely loose. As a result, it appears that it will become easier to track and control changes made to functions, reducing the production of unwanted side effects across the rest of the system. Additionally, the scalability claims associated with the paradigm have the potential to help us solve some of the challenges with respect to model performance.

The key benefits of the paradigm have been hypothesised by Sousa and Ferreira (2012), which stands as a key paper in summarising the current state of the art of this paradigm. However, there is very little evidence to suggest that these hypothesised

benefits have been explored further. In particular the capability for Object-Functional to eliminate the anti-patterns associated with object-orientation, which is a key issue for the MODs extant campaign model range. An evaluation of the Object-Functional approach applied to defence models would provide an indication as to whether this is the case. However, whilst the goal of this research should be to understand further the potential benefits and challenges of exploiting Object-Functional, for the purposes of developing an OA model; the application of this paradigm via the GAMOV framework, including any key lessons learnt would also contribute to the growing body of knowledge in terms of formulating sets of software design patterns for the paradigm.

3 Research Methodology

3.1 Chapter Introduction

So far, this thesis has examined the overall challenges facing defence with respect to the development of their software models and has examined the literature concerning the state of the art of Object-Functional software development. This next chapter describes how the research questions of this thesis shall be answered by outlining the research methodology and the methods that were employed to understand more about Object-Functional model development and how it can potentially benefit over other approaches, such as Object-Orientation that were used in extant capabilities.

3.2 Approach

As per the research questions, the goal of this thesis is to understand more about the application of Object-Functional programming to the development of defence OA software models, specifically the benefits and challenges it may offer; and what such an approach could look like, which shall be presented in chapter 4 in the outline of the GAMOV framework.

In order to demonstrate potential benefits and challenges of the Object-Functional approach, it is important to compare and contrast extant models produced using purely Object-Oriented and/or other techniques with those produced using Object-Functional. The remainder of this thesis shall therefore evaluate a set of models produced using a range of these approaches and attempt to understand more about the underpinning structures offered and what they potentially allow analysts and model developers to do.

The majority of the data capture and analysis was qualitative rather than quantitative. This is because the research is examining patterns of development and coding practices being undertaken within Dstl, which are fundamentally qualitative processes. Whilst quantitative analysis of software can reveal interesting insights into specific design approaches, this research is looking at the overall impact of implementation practices upon a model’s ability to deliver analytical quality.

Additionally, a quantitative approach for this particular piece of research would not have been viable for a number of reasons:

- Software Metrics: As Dstl's long standing focus has been on delivering analytical quality, rather than software performance, there is a lack of key performance indicators or metrics logged for things such as model runtime, algorithmic performance etc. Such information is only available through discussion with expert users of the model by noting their experiences and statements recorded within documentation.

- Model Variation: As the models cover different areas of analytical interest and vary greatly in terms of complexity of representation, it is not possible to design a single quantitative experiment that all evaluated selected models in the set could process.

- Training: Given the complexity of these models and the required read-in to become proficient at setting up a scenario, the level of work to set up a quantitative experiment outweighed the benefit. Whilst the process of setting up a scenario in the model may have personally exposed some of the underlying problems with the code, it was more effective to collate this experience from an expert who has been through the process themselves and analysing what they have empirically observed. Additionally, the experts view would be more comprehensive as they will have used the model for many different purposes over many years.

3.3 Choice of Models

In order to choose the models used in this evaluation exercise, identification of the selection criteria was important for their selection and any constraints.

3.3.1 Model Coverage

The Aqua book (HM Treasury, 2015), which is the U.K governments primary guidance on delivering analytical quality, recognises eight categories of model used in the creation of analytical products. These are listed on page 15 as follows, with defence examples given below for clarity:

- Policy Simulation: Understanding the implications of current defence policy on our ability to prosecute various operations around the world.

- Forecasting: Some defence models are typically used to forecast outcomes of employing certain tactics or strategies and to assess the potential of using different military capabilities within a scenario.

- Financial Evaluation: Some defence models are produced in order to understand the impact of investing in various defence capabilities and military tasks.

- Procurement and Commercial Evaluation: Based upon analysis conducted in a multitude of areas, defence models provide advice in the decision making of what military capabilities to procure.

- Planning: Defence models are often used to understand the impact of employing different courses of action in order to achieve a scenario outcome.

- Science Based: In defence OA, these are typically modelling of physical systems, which are often subsystems of a larger campaign model when used in DSA analysis activities.

- Allocation of Funds: Whilst there are models and business units within Dstl that undertake these activities; defences models within DSA do not typically provide direct input to this capability. However, the outputs from other capability assessments may support some decision making in this area indirectly.

- Conceptual: As per the Aqua Book (HM Treasury, 2015, p.15) - *"to help understand the key influences that are important to a system being modelled."* This is an important analysis activity in order to understand more about the defence environment and uncover new insight worthy of further analysis.

Going by these criteria, the models picked for this evaluation aimed to cover as much of this spectrum as possible, so that all categories of model are included in the analysis. However, in defence, whilst there are examples of specific models covering a single category in this list; in actuality many of the larger models cover multiple categories. For example, a campaign level model is capable of being used for simulating policy, forecasting demand, planning of operations, understanding science behind various systems and aspects of conceptual modelling, all within one

simulation system. However, campaign models rarely evaluate aspects of finance; therefore, another model specifically designed for this purpose had to be included.

3.3.2 Implementation Type

It was decided that the choice of models should cover as many types of development commonly undertaken within Dstl, in order to get a flavour of all development activities. This typically includes spreadsheet models, models based entirely or partially on commercial products, models based upon frameworks and bespoke solutions.

3.3.3 Final Choice of Models

Taking the two criteria of model coverage and implementation into account, the following models were picked for further evaluation:

- Wartime Planning Tool (WPT): A spreadsheet model used for representing individual combat engagements.
- Strategic Balance of Investment (StratBOI): A model partially based upon commercial software used for analysing financial investment in military capabilities.
- Diplomatic and Military Operations in a Non-war fighting Domain (DIAMOND): A framework derived, high-level campaign model, built using Object-Orientation.
- COMAND: A bespoke, high-level campaign model, built using Object-Orientation (as described previously in chapter 1).
- Aerial Delivery Model (ADM): An Object-Functional model built using the GAMOV approach.
- Mission Command Model (MCM): An Object-Functional model built using the GAMOV approach.

3.4 Evaluation Framework

Research into the range of models and frameworks that can be applied to the evaluation of software quality has been undertaken as far back as software has been developed. Research conducted by Miguel, Mauricio and Rodriguez (2014)

summarises many of these dating back to the 1970's until present day and the criteria commonly used throughout. Many of these criteria have now become standardised under ISO25010:2011 by the International Organisation for Standardisation (ISO). (2011); however, Miguel, Mauricio and Rodriguez (2014) notes that some important methodological criteria such as communication and open source community practices are not explicitly covered by this standard.

With respect to modelling undertaken within government departments, including defence, the Aqua book (HM Treasury, 2015) is the principle piece of guidance in terms of encouraging best practice and delivering for analytical quality. The core theme of the Aqua book with respect to models is based primarily on how the model conforms to the principles of RIGOUR (Repeatable, Independent, Grounded-in-reality, Objective, management of Uncertainty and Robust). This research is examining how the implementation of various models impacts their ability to deliver analytical products under the analytical process, therefore evaluating them should be in the light of those criteria the Aqua book considers to be fundamental. However, as these models are software products, criteria for ISO25010:2011 shall be used as inspiration for compiling an assessment framework for this research.

ISO25010:2011, divides evaluation criteria to be considered for the evaluation of software down into eight categories, which includes "Functional Suitability", "Performance Efficiency", "Compatibility", "Usability", "Reliability", "Security", "Maintainability" and "Portability". Not all of these categories are relevant for the evaluation being undertaken within this research. The relevance of each category was assessed as follows:

- Functional Suitability – This encompasses criteria with respect to completeness, correctness and appropriateness of the software for the purpose for which it was designed. This is relevant to the research in terms of framing how the issues with the implementation of the software impact its functional suitability, which is also what the Aqua Book (HM Treasury, 2015) is concerned with preserving. However, this research is not answering the overall question of whether the model is fit-for-purpose, to which the answer is already predefined by the model being in usage or not on studies.

- Performance Efficiency – This encompasses criteria with respect to underlying performance of the individual systems. This has some relevance to the overall assessment, because scalability and the improvement of algorithmic performance via parallel processing techniques is a key goal for Dstl. Therefore, assessing how the model's implementation allows and restricts its capability to exploit these is of importance. However, lower level performance metrics do not exist for these models; and even if they did, they would likely be rough orders of magnitude based upon the expert user's experiences.

- Compatibility – This an assessment of how the software is able to interact with other products within its operating environment. This is not relevant to this assessment, because of the bespoke nature of these models and other external methodological factors associated within the organisation.

- Usability – This category covers a range of criteria from training and operability, user interface considerations and accessibility. Some of these will be relevant to this research, particularly operability and the user-interface; because issues identified here may have some relationship to the implementation. Training with respect to defence models is less focussed on understanding the software and more concerned with training of producing a valid representation within the model. Accessibility, whilst important in a wider sense of the user experience, is not important for this research.

- Reliability – Much of the criteria with respect to reliability is externally managed through Dstl's commitment to achieving ISO27001 via its accreditation. Therefore, criteria under this category with respect to maturity, availability and recoverability, whilst possible side-effects of the software are typically a product of wider methodological and environmental factors that are neither relevant nor suitable for comment within this research. However, an overall question with respect to the reliability of the models could be asked to uncover issues with respect to fault tolerance and how the models handle things such as error trapping.

- Security – This category overall is not relevant to this research. Whilst arguably software implementation could present issues for this category, the overall fitness-for-purpose regarding security is not relevant.

- Maintainability – This is a very key category of criteria for this assessment, because it includes aspects such as how much modularity exists within the software, how reusable are the components and to what degree of testing is present.

- Portability – Criteria here covers aspects of installation and adaptability to new operating environments. Much of this is not relevant to the research because this would likely arise from things such as the programming language choice.

In addition to these criteria, it was key to record explicit details with respect to the overall implementation, for example, what paradigm is the code-base following, what is the overall size in terms of lines of code etc.

Taking all of these criteria into account, the following evaluation framework was developed in order to assess the models within this research:

Category of Criteria or Question	**Questions to be asked.**
Functional Suitability	What is the purpose of the model? – *What types of study and analytical work is the model employed on* What categories of analysis capability does this model cover? – *As per the Aqua Book* (HM Treasury, 2015, p.15)*.* Is the model currently in use? - *If not, for what reason(s) was it retired or replaced by another capability? Were any of these due to the implementation?*
Overall Design Questions	What language and/or coding environment is the model written in? What coding paradigm is the model stated to follow (or appears to follow upon inspection)? – *Object Oriented, Object-Functional etc.* How large is the code-base? – *Lines of code.*

	Are there any observable anti-patterns? – *From looking at the code-base, or from expert's view.*
Performance Efficiency	Is the model scalable? What scalability approaches does the model currently employ (if any)?
Usability	What options does the user have in order to interface with the model? – *Graphical User Interface, data files etc.* Typically, what options do users use? – *Do user(s) use the provided approach or have they developed their own method?*
Reliability	Are there any observed or documented reliability issues of note associated with this model?
Maintainability	What level of modularity does the software employ? – *By function, by functionality area, by system etc.* What approaches to testing does the model employ? – *Unit, Integration etc.* How reusable are the components within the system – *Based upon the expert's experience.*

Table [1] – Model Evaluation Criteria

3.5 Data Capture

In order to populate these questions, a number of qualitative techniques shall be used. As stated previously, most of this information shall come from discussion with the expert users of the model, which in Dstl are known as 'custodians' of the model; and recording a summary of their expert viewpoint and empirical observations. It

should be noted that not all custodians are necessarily software developers and may not have hands-on experience of using the underlying code. However, the custodian is the primary point of contact and thus would be aware of issues surrounding the implementation through collation of lessons learnt from development activities.

Another source of information that shall be drawn upon shall be the model's documentation, in particular, the model logbooks which are non-publishable, internal documents held at Dstl that track aspects such as development history and outcomes.

Lastly, where applicable, the code-base of the models shall be analysed by manual inspection to see if observations with respect to its structure yield any insights in support of the questions. This is the process known as 'code-smell' which is summarised in works such as Fontana, Zanoni, Mariona, and Mantyla (2013). The principle behind code-smell is that a developer can identify potential issues and anti-patterns from the code-base by observing aspects such as structure and modularity. Whilst automated code-smelling techniques are actively being developed as per Fontana et al. (2013), because this technique is prone to subjectivisms, the techniques shall give a sense of what may be occurring within the models combined with the model expert's observations.

4 The GAMOV Approach to Object-Functional

4.1 Chapter Introduction

In order to further inform the analysis from the evaluation, this chapter shall outline the design of the GAMOV framework used to produce the ADM replacement and MCM models; defining its purpose in the context of what it is trying to deliver and the layers of capability that are contained within it. The design of the lower level aspects of the framework shall then be covered, outlining the individual subsystems that make up the framework's architecture and the interactions between these subsystems. This shall help in developing the understanding of how GAMOVs interpretation of the Object-Functional approach enables it to produce a much more flexible modelling solution and shall contribute as a possible approach for Object-Functional application in the field.

4.2 The GAMOV Framework

In Chapter 1, a number of key issues with respect to Dstl's campaign modelling capability were highlighted. In an attempt to address these identified issues, the concept of the GAMOV modelling framework was developed (Glover & Toomey, 2012). It was developed iteratively over 9 years spanning late 2008 to early 2017 by a team that varied between 3 - 4 developers, including the author of this thesis who was a leading member, particularly with respect to owning the conceptual design.

GAMOV was envisioned to be a modelling framework, containing within it a set of reusable components, with an associated API, to enable modellers to build models from its libraries of components. In that sense, the capability of GAMOV would be comparable to a programming framework, such as the Microsoft .NET framework, albeit with a much more defined context and purpose in mind.

The final architecture and organisation of the GAMOV software was framed around the characteristics brought about by the Object-Functional paradigm, which we believe helps to promote the reuse of new and existing ideas and provide future resilience for modelling within Dstl. However, the formal acknowledgement of Object-Functional did not occur till much later in the actual development cycle. As stated in the literature review, because REST is believed to share many of the characteristics of Object-Functional as it is currently understood, it was in fact that protocol that

initially drove the design of GAMOV, at least in the early stages between 2008 and 2012. Further details of this observation were presented in (Boakes & Toomey, 2012). As further development of the framework intersected with this research, it was identified that Object-Functional, rather than REST, was the more appropriate conceptual basis for the framework. However, due to the strong similarities between REST and the interpretation this research had made of Object-Functional, the GAMOV code-base did not require any modification to its design to become compliant with the concepts of Object-Functional.

The name 'GAMOV' is in acknowledgement of the works by the Russian theoretical physicist George Gamow[2] ("George Gamow Biography", n.d.), who amongst his many accomplishments was renowned for simplifying the description of very complex scientific concepts into terminology that children could understand. This, in a way, was comparable to the ambition for GAMOV: to simplify the approach to the implementation of models, so that the modeller is empowered with a flexible system to explore their ideas.

The goals of the GAMOV framework were thus:

1. Increase the agility of campaign model implementation, by providing highly reusable, 'off-the-shelf' modelling components:

    a. It is believed that the exploitation of the Object-Functional paradigm eventually became a fundamental enabler of this goal. The separation of data from functionality using highly reusable interfaces enables an approach akin to 'plug-and-play'. Caution was taken not to describe this approach as true plug-and-play for a number of reasons:

        (1) Firstly, the plug-and-play descriptor risks implying a more simplified approach than in actuality to a non-modeller, or potential customers of GAMOV. This may risk trivialising the approach and positioning GAMOV into providing support to projects in unrealistic timeframes.

        (2) Secondly, it is important not want to mislead modellers using GAMOV into thinking that they did not need to concern themselves

[2] 'Gamov' is the Russian pronunciation of Gamow

with the wider implications of the components they are plugging together. Whilst the increased flexibility of this approach would aid aspects such as validation; it risks at the same time in aiding in the production of an equally invalid representation without a sufficient level of diligence from the modeller. This was felt to be important aspect of this approach for the modeller to have some explicit engagement with the interfaces between components so that they are aware of the implications of their decisions with respect to the validation of their representation.

2. Provide an environment that will enable bespoke models to be built for the purposes of the intended study. The hope was that this would move the resultant models away from the current state of the art, whereby representations of scenarios are being shoehorned into a pre-existing model.

3. Enable an iterative approach to model development, where model verification and validation become an intrinsic part of the process of the production of a model, rather than a task that is performed post-production and help manage aspects like uncertainty better as per the ambitions of the Aqua book (HM Treasury, 2015):

    a. This is colloquially referred to as ‘model-test-model’ by the development team, whereby components can be added to a model (or removed from a model) iteratively. The impact of that addition (or removal) can then be tested and evaluated, resulting in stronger direction of how to achieve overall validity of the model.

    b. Exploitation of modern software development approaches was therefore key, so that testing and documentation are part of the code-base in order to make these activities as agile and repeatable as possible. As stated previously, this was through adoption of REST (Fielding, 2000) and then Object-Functional.

Given the outcomes of the ‘McPherson’ review (McPherson, 2013), a greater emphasis is rightly expected, in order to ensure effective quality assurance with respect to the modelling that is undertaken and recordable at all stages of a

model's development cycle. GAMOV aids in this regard by making the process of V&V intrinsic to the core GAMOV philosophy of building a model.

Through realising these goals, it was intended for the GAMOV framework to provide benefit to the overall analytical process in terms of savings to both time and cost incurred on studies. This was very much a long-term ambition from using the GAMOV approach and could only be realised through mass adoption and continued usage of the framework on studies. As the number of available components and models produced using the GAMOV framework grows, the potential to build new models through adaptation and repurposing of pre-existing models and their components would become more commonplace, as opposed to building every new model from scratch.

It is important to make the distinction that GAMOV was not an attempt to repeat the production of the DROMAS approach. Whilst there are undoubtedly strong similarities in the overall goals of the approach, there is a distinct difference. GAMOV is not focussed on providing models that are consistent in terms of their framework. Whilst this idea of DROMAS was to simplify the construction of models and provide a familiar operating environment, it constrained the solution space by constraining the model's overall framework. Thus, resultant models became slight variations on a theme. GAMOV sought to empower modellers to rapidly define both their model framework and the component organisation within that framework.

4.3 The GAMOV Layers

As GAMOV is a framework, there are a number of layers of capability present in order to provision the overall software. Some of these layers are not part of what constitutes 'GAMOV' with respect to the modelling software itself. These are enablers in terms of the underpinning hardware and software to support its operation. However, it is important to conceptualise these as part of the overall GAMOV capability, in order to understand the relationships and dependencies upon the development and usage framework.

GAMOV can be conceptualised using the following diagram (See Figure 4).

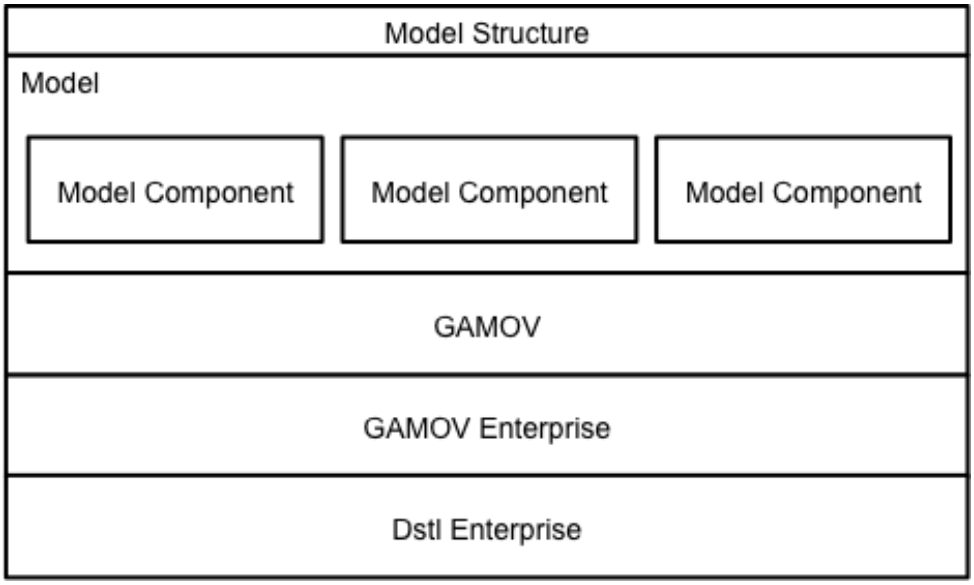

Figure [4] – The GAMOV Layers

Starting from the bottom-up, the function of each of these layers shall now be outlined in the next few subsections.

4.3.1 Dstl Enterprise

This incorporates the unmodified hardware and software that is provisioned by Dstl Information Communication Technology (ICT) infrastructure in order to run GAMOV. At a high level this includes the machine specifications (be it physical or virtual), the operating systems, networking configurations, programming languages and supporting tools, such as configuration management.

4.3.2 GAMOV Enterprise

This layer is composed of the additions or modifications that have been made to the Dstl Enterprise by the GAMOV development team in order to run GAMOV. Examples of this include additional libraries for programming languages, Integrated Development Environment's (IDE), web server configurations etc. In other words, this layer is composed of those elements that can be controlled and defined by GAMOV developers.

4.3.3 GAMOV Layer

This is the GAMOV software itself, and all of those components of GAMOV that are common across all models. For example, the critical subsystems that runs a GAMOV model.

4.3.4 Model Layer

This layer is the organisation of model components that is specific to a particular model implementation via the Object-Functional organisation. Every component within this layer varies in reusability between independent functions that can be reused by all models ever built using GAMOV, to specific configurations of components that can only be reused between families of model. For example, a routine for moving entities around the environment could potentially be reused by all entities in any model, whereas a system for processing logistics is more specialised to logistics-based models.

4.3.5 Model Structure

This is a wrapper as opposed to a layer, but the conceptualisation here is to signify that there will be some element required in order to package each version of a model for release into usage by analysts (i.e. a model specification). In terms of implementation, there is typically a base class for the model that outlines this.

4.4 GAMOV Approach to Model Construction

4.4.1 Data and Functionality Separation

The core idea within GAMOV is that the actors and the functions were to be separate objects, and were conceptualised using two data structures as per Object-Functional. One of these contains the data elements and the other contains the functionality required of the model representations. The first is the ‘Entity’, which is used to represent every actor within the simulation. The second is the ‘Mediator’, which is interface for the transformative functions that shall be used upon the entities and similar to the naming convention used by Kontio, Mäyrä and Rönkkö (2007). This was initially designed around the notion of ‘client’ and ‘server’ as outlined within the REST protocol (Fielding, 2000), whereby clients are serviced by remotely-called functions. As the existence of the Object-Functional paradigm became acknowledged

in the later stages of development, the key concepts remained the same, but merely the nomenclature would change from ‘clients’ and ‘servers’ to data objects and function objects (or functional classes).

4.4.2 Entities

4.4.2.1 Entity Conceptual Design

The Entity structure in GAMOV can be used to represent every actor within a model. An actor in GAMOV terms is anything used to build the representation of a scenario in terms of data. This would therefore include, at a high level, all of the actual force elements (the troops, tanks, planes, ships etc.) and elements of the environment such as locations. The GAMOV entities were also designed to inherit the same data structure. This means that every entity within a GAMOV model is fundamentally the same in terms of structure, even though they are representing completely different ideas.

This approach to genericising the data structure is not characteristic of the Object-Functional paradigm and is a specific characteristic of GAMOV. The main driver for this approach was to have the ability to rapidly realise new military concepts using one data structure rather than creating an entirely new object class every time in order to mitigate the issues we had experienced with respect to highly coupled code. Additionally, this design for entities allows for entity descriptors to be reused or adapted quickly rather than producing a new one through the data.

A historic problem that the team were cognisant of from our collective experiences from maintaining older modelling capabilities is that when a new military concept had been devised, there seemed to be a tendency by previous model developer(s) to add these to the model in the form of its own object class. The problem with this is twofold:

1. In many cases the modeller is simply adding a subtle variation of a pre-existing capability that is already present within the model. This produces extra overhead in terms of managing the interactions associated with this new object class, but also there is a duplication of conceptual representation occurring. For example, there is no need to produce completely different objects in order to represent a frigate vs. a destroyer. In principal whilst these two vessels operate in fundamentally different roles in the maritime domain, they possess many of

the same data attributes such that a single class of object could describe both of them. Their capability within their role can then be defined through changes to these data attributes.

2. Because these entities are so similar, coupling is produced because they are exploiting one another's functionality as explained earlier by Ottinger and Langr (2011).

Using the same data structure for every actor means that new data attributes can be added simply by inserting a new entry into the entity data structure. This enables new concepts to be realised rapidly, or enable pre-existing ones to be modified with very few changes to the code. For example, if the requirement for a pre-existing land vehicle to become amphibious emerged, this could be achieved by setting the entity's ability to move through water to a non-null state, rather than declaring an entirely separate entity class for this subtle change.

A key characteristic of the GAMOV entity design is that entities can hold other entities, which is similar to the proposal brought forward by Jalote (1989) in his proposed extensions to the object-oriented paradigm. This is what enables the representation of military force hierarchies, but also the representation of complex systems. For example, an entity representing a physical location will hold all of the entities representing the military or civilian units that are stationed there. These unit entities may hold other systems such as weapons that may also be represented as entities. This is also similar to the ideas of entity composition as presented by Odersky (2014) in terms of the Scala language, however, the entities are not being fused via a composition method; simply the entity reference is held in an attribute container.

However, for all the flexibility afforded by this structure, the transparency of what an entity is in the context of the code is lost through this approach. In Object-Orientation, the class structure of attributes and methods can help to illustrate what the real-world context of the entity is representing. Under the GAMOV entity approach, every entity is the same, with its capability defined by what attributes have values and those that are set to a null state. In this way, an entity in GAMOV terms is characterised by what functionality it can make use rather than what functionality it owns.

4.4.2.2 Entity Implementation

In terms of implementation detail, the entity uses a variation of Python's built-in dictionary structure, known as the 'Mutable-Mapping' class to give the entities a hierarchical organisation. The Mutable-Mapping class is then inherited by a bespoke class know as a Fixed-Structure dictionary (known as a 'FSDict' for short). The purpose of this class is to provide the requisite control mechanisms for the data structure and to ensure that the modeller cannot change the base structure of base GAMOV entity, either directly or indirectly.

As stated earlier in section [4.4.2.1], whilst generic entities in this fashion are not an explicit characteristic of the Object-Functional paradigm conceptually speaking; by having a generic interface between these data items and transformative functions was found to be key in terms of implementation in order to preserve the interfaces in the overall system. Having multiple entities with fundamentally different base classes could limit the reuse of transformative functions, or said transformative functions would require the capability to operate through multiple interfaces. Therefore, ensuring that all entities are truly identical in terms of their underlying structure for all models was key, which is why these constraints exist. This also a key characteristic of the REST protocol (Fielding, 2000).

In cooperation with Fixed Structure dictionaries, the ability to freeze a dictionary's contents via the FrzDict.py class is also available. This was added to provide an optional feature for those times where the modeller may wish to ensure that the data of an entity cannot be modified, which may be useful to preserve baseline assumptions when sharing the model with other analysts on a wider study.

So far, everything described about the GAMOV entity implementation has been concerned with structure. The attributes for GAMOV entities are then defined within this structure using the Entity.py class. These are the generic attributes that are common across all GAMOV entities within any model produced in the framework. For example, all entities will have some form of movement score, an attack capability and a damage score to name but a few. This structure is then inherited and extended by the individual model, using a specific entity class for each model, in order to add those attributes that are only common to a particular model or family of models.

Finally, with respect to Entities, the concept of a Force Element container exists, represented using the FrcEle.py class. This is used to logically group entities together into force structures, which are required for certain mediators that work at a higher level of aggregation. For example, if the model is representing entities in one hierarchical unit (i.e. a company or a division etc.) but needs to group them together to represent a more complex structure (i.e. a battle-group), this structure aids in this regard. This grouping is different to that of how entities can hold other entities and does not affect that relationship in any way.

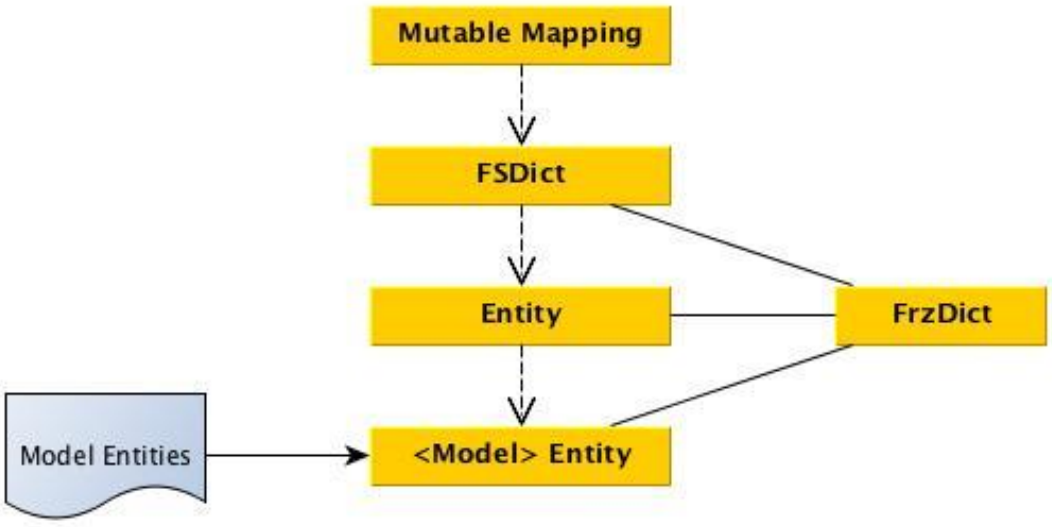

Figure [5] – Entity Structure

As flexible as this structure is, there is risk of defining attributes at the wrong level of this entity hierarchy. For example, generic attributes that should belong in the generic entity used by all GAMOV model could be defined in model specific entities. This has to be controlled and assessed on a per-model basis as a matter of course, in order to determine whether attributes are generic enough in nature to warrant their relocation higher up the hierarchy. This presents a potential configuration overhead that users need to be aware of when using the framework. With this current structure, changes made to the values in generic entity level have the potential to affect the model specific entities derived from this class. However, at the same time, by keeping generic functionality encapsulated too far down the structure could lead to duplication of concepts through data.

4.4.2.3 Entities for Representing Environment

As stated previously in section [4.4.2.1], all actors within the model are represented using the entity data structure, which conceptually included nodes in the environment, but not arcs. Typically arcs would be no more than a logical linkage between two node entities within the data, in order to allow the movement of entities across the battle-space. However, there are defence models in existence that conceptualise arcs as something that can be attacked. Not only that, there are also models that breaks the arc concept down into positioning along an arc, whereby certain portions along an arc can be degraded. This is to allow for the representation of bridges being destroyed or the production of impassable terrain as a result of an explosions or the presence of a threat. Therefore, arcs also had to be conceptualised and implemented as groups of entities within the GAMOV framework in order for the transformative functions to have an impact upon them. Making this change provided more consistency for the conceptualisation of a model in terms of Object-Functional, because now everything that is data within the representation is now truly an entity. Arcs are not actors in the classic sense of a model, but in terms of how GAMOV conceptualises actors in light of Object-Functional design, they are data elements of the representation, thus permissible as Entities. The risk here is that this is a non-classical approach in terms of other modelling domains where arcs are merely logical linkages. This will have to be managed in terms of learning and transparency of the code. Other modellers will likely assume no more of an arc than a logical link, and would not expect it to be an entity in its own right.

4.4.2.4 How Entities deviate from Object-Functional

So far, everything described about the implementation follows the characteristics of Object-Functional as understood. However, the GAMOV entity does contain one minor and necessary deviation, which is the requirement to hold a single piece of transformative functionality within the entities themselves.

Not all of the functionality that entities would be interfacing with would necessarily be using a consistent scoring system for an entity's attributes. For example, in defence modelling there is more than one scoring system used to represent damage that would be inflicted upon an entity. This meant that there needed an attribute for each of those scoring systems in the entity hierarchy and a method of ensuring that when

one of these attributes was modified, the equivalent scoring systems were also modified to maintain equality.

The entity was therefore given what is colloquially known as a ‘book keeping’ methods, which will be routinely called after an entity transformation to ensure that other scoring systems are updated to their equivalent value. On the surface, this could be argued as a break in the design of conceptual structure of Object-Functional, at least in the purest sense of the application of the paradigm, because now there are methods being held by entities themselves. Whilst this is true in terms of component organisation, it was not a break in the paradigm in terms of the production of side effects. Given that this book keeping functionality was only modifying the state of the entity that calls the function, it is not going to produce side effects upon any other entities within the system. This could be argued as a permissible deviation from Object-Functional; because whilst the book keeping methods are transformative they are only transformative to the specific data entity that called them and no others in the simulation.

4.4.3 GAMOV Mediators (Transformative Functions)

The mediator is the container and the interface for the transformative functions used in models. Mediators are stateless as per the principals of both REST (Fielding, 2000) and Object-Functional design; meaning that they are agnostic to the specific entities that they are performing their operations upon. Function calls to specific mediators shall be scheduled as events to occur either at prescribed times or as a result of another event in the model. Depending on its purpose, the mediator will perform its function upon any entity that is in the correct state. For example, if a combat mediator is running, it will in the case of two-sided combat look for all instances where a Blue (friendly) entity and a Red (enemy) entity that are deemed as being in combat, and then perform a combat calculation on all of these entities. This is similar in concept to the Simkit approach (Buss, 2002) that uses event listeners to trigger the correct sequencing of events; however, the GAMOV implementation is much simpler in that it performs a scan across the environment locations for instances of combat.

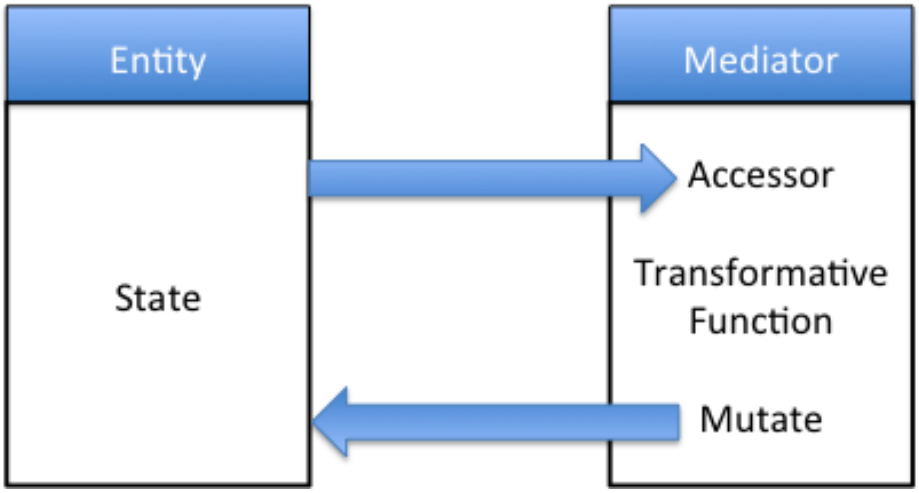

Figure [6] – GAMOV Entity and Mediator Interactions.

The conceptual structure of a mediator uses a three-stage process in order to transform an entity's attributes:

1. Accessor Function - This is the stage where an entity's data is fed into the mediator and pre-processed. In terms of pre-processing, this only concerns the logical order and assigning of values to the parameters expected by the mediator. This is not a transformative stage in any way.

2. Transformation Function - This would be the transformative function contained within the mediator itself; in other words, the mediator's namesake.

3. Mutate Function - This stage takes the outputs of the transformative function and reflects the result back onto the entity structure. This will also schedule calls to those book keeping routine mentioned in section [4.4.2.4] in order to keep entity attributes in sync.

4.4.3.1 Mediator Implementation

Whilst this three-stage concept of the mediator looks complex in terms of the design, the implementation in terms of code is actually very simple, provided the interface to the entity is consistent for all mediators, further emphasising the requirement for all entities to have a consistent base structure. However, it is also very easy to break this three-stage process. For example, it could be easy for an inexperienced developer to couple the behaviour of the accessor function within the transformation

function, whereas this needs to be kept separate to preserve the clarity of the interface. As a result, there is a lot more requirement for conceptual control that cannot be enforced by coding constraints.

At least in terms of GAMOV, this is an important consideration for preserving its Object-Functional design. Frequent review and control of the implementation of mediators would need to become a regular task by anyone who is building a model.

4.5 Other GAMOV Subsystems

So far, the high-level structure of the GAMOV framework has been described in terms of layers and the organisation of data and transformative functions via the entity and mediator concepts. However, the GAMOV layer (illustrated within figure [4]) is broken down into a number of other subsystems.

4.5.1 The GAMOV Engine

The GAMOV Engine is effectively the kernel of a GAMOV model that drives all of the processing and schedule all of the interactions between components. The GAMOV engine contains those components that define the architecture of all models produced using the framework.

The engine contains the following subsystems:

- Event Scheduling – The mechanism by which events are scheduled and kept in sync during a model's operation;
- Entity – The base data structure that shall be used to describe all actors within the model;
- Run Object – This is the main thread of execution in order to run a GAMOV model. This aspect is also part of what defines the 'model structure' as shown in figure [4]. Each model has a Run Object that instantiates all of the components required for a particular model; hence it defines the structure. However, as a construct as part of the implementation, the Run Object is actually an engine component.

4.5.2 Event and Time Management

At the core of the event management system is the GAMOV clock, which is used to granulise time values and ensure that events are dispatched at the correct time and in the correct sequence. Fundamentally, all models produced from GAMOV are event driven, but the clock implements mechanisms designed to allow for time-step intervals, for the scheduling of events that are characteristic of time-stepped models.

The GAMOV clock automatically granulises all time values into the same time units that are defined by the base time-step. This enables model designers to be flexible in how they define their time values within their inputs, whilst having the assurance that a conflict will not arise during execution as a result of inconsistent units. Additionally, this spares the end-user the requirement to pre-process existing data that may not be in the standard time units that the model is designed to use. Typically, many of DSAs campaign models runs in seconds, but finer granulation of time may be required if for example a mechanical system was being represented. Therefore, the clock was implemented in such a way as to allow the submission of these time units whilst ensuring their automatic management in order to provide a consistent view of time across the model.

The clock monitors all of the events stored within the event scheduler, which is composed of a variety of queue types. The ‘event scheduler’ is a colloquially known concept within the GAMOV, but in actuality the queues are attributes of the Clock itself. The GAMOV Clock has the following queue types in order to satisfy the whole range of modelling event types:

- Immediate Queue – This is the highest priority queue within the event scheduler, which is designed to contain events that must be processed before all other events in the model. In OA terms this would typically include events to synchronise model activity and ensure that certain routines have completed before proceeding with further events. An example of this would be the book keeping routines as described in section [4.4.2.4], as the model should not process any other action until all scoring values for the entities are consistent with one another;

- Cyclic Queues – Allow for the scheduling of events to occur on a pre-defined cycle (for example, every 12 hours). This queue is used for the scheduling of the planning routines and update cycles that commonly occur within the Command, Control, Communication and Computer (C4) Intelligence, Surveillance and Recognition (ISR) systems (or C4ISR as it is known), because in reality commanders typically plan and update operations on a defined cycle;

- Time-Step Queue – Used for the scheduling of jobs to occur on the defined time-step of the model. This queue handles all model events that are naturally modelled on a time-step;

- Main Event Queue – Used for scheduling one-off events that occur at a specific timestamp in the model.

It is permissible for GAMOV queues to contain other queues of events, if the user requires a separate sequence of events to occur at a specific time. It is also possible to create more than one clock if required, however, typically this is not something that was envisaged to be commonplace.

Lastly, the GAMOV clock is currently designed around the concept of an end-to-end simulation, not a war game with a human-in-the-loop. If in the future, human-in-the-loop decision-making was required, then it may be necessary to create a new queue type in order to interrupt the model when the user wishes, in order to change the state of entities and orders as part of a war game.

4.5.3 GAMOV Engine Implementation

In order to have queues that are event-driven, queues are distinguished by priority. At the time this part of GAMOV was implemented, which was early 2009, the Python language only provided the capability to define the priority of the events on a single queue. It did not allow for the queues themselves to be assigned a priority. Therefore, a new class was created, known as ‘Priority’, whereby the standard Python ‘Queue’ object could have an associated priority. This enables the concept of an immediate queue, which has a priority of 0. As to the function calls to the mediators themselves, these would have to be represented as first-class objects in Python, which is the functional class equivalent for Object-Functional, so that they can be stored within the queues.

To ensure that time is granulised across all events stored on the Clock, the ‘GTime’ class was created. Typically, there shall only ever be one GTime object produced per GAMOV model, but it is the object’s responsibility to ensure that all values of time are granulised against the pre-defined base time step interval that is being used by the model.

As events are brought to the front of their respective queue, they are dispatched using the ‘Dispatcher’ Object. This object does nothing of significant note, other than to execute the function calls to associated mediators or other routine that would be scheduled on one of the numerous queues. The final implementation of the GAMOV time and event management subsystem can be summarised by Figure [7].

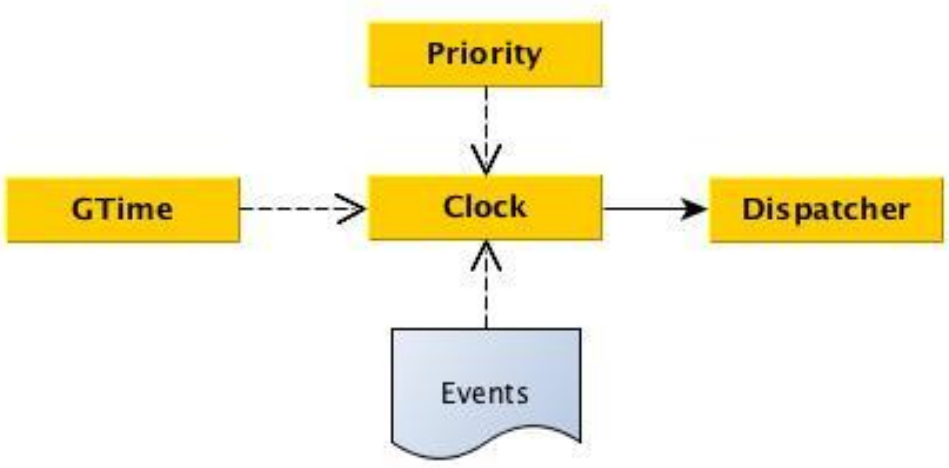

Figure [7] – GAMOV Time Management System

4.5.4 GAMOV Engine and Object-Functional Design

Overall the design and implementation of the GAMOV engine, on the surface, is not integral to Object-Functional, but key features are believed to provide enablers to at least the ‘Functional’ side of the paradigm. For example, the ability to schedule mediator calls as ‘first class’ objects on the queues enables functions to be scheduled when they are supposed to happen and the system allows for the correct sequencing of mediators to be explicitly defined rather than leaving it implicit within the structure of the code.

There is however, one aspect of this design that risks breaking the Object-Functional paradigm. In order to track the total number of clocks being used within a model, a global variable is produced. This global state is required to ensure that new clocks are produced with different names in order to prevent conflicts. This state is not used by the operations of any other subsystem, only the clock, in order to restrict the spread of state across the framework. Avoiding global is key to many paradigms, not just Object-Functional, but this is very much a controlled exception viewed through the lens of not producing unintended side-effects.

4.6 Configuration Management System

The intention of the GAMOV configuration management system is to provide all the components required in order to setup a GAMOV model in terms of data input, including the run-set configuration; and process the output of the model into useful log files.

4.6.1 Input

Inputs to a GAMOV model are comprised of the following:

- Entity setup – The user is able to define the data values of the Entity template, but also customise the template with different fields for the purposes of the model.

- Replications - The user is able to define the number of replications that they would like to perform as part of the run-set and the associated configuration data for each of those replications e.g. what random number seeds they will be using.

- Order set – In the instances where scripted command and control is required, there is a mechanism by which the scripted actions can be submitted that will drive the representation of decision making within the model.

There are two important features that were designed into the GAMOV input system in order to provide improvements over older campaign level models:

1. First is the ability to merge data files. This allows the modeller to be able to separate input data into as many (or as little) files as needed. This is designed

to aid the modeller in logically structuring their data towards the requirements of the study (and arguably their personal preferences), but also to provide extra transparency to future modellers, who may be reusing a model, so that they can quickly identify how the data is organised.

2. Secondly, the ability for a modeller to declare changes to the baseline data through a 'change file'. In practice, when using a model on a study, there is often the requirement to explore variations to the baseline scenario (the 'what if' questions) where small changes to the data are made. This can be as small as increasing or decreasing the number of available assets or modifying the performance of capability; or as large as changing the entire script of events.

4.6.2 Output

Traditionally in other OA models within DSA, the format of the outputs has been predefined within the code. In other words, the model produces very specific log files for common areas of analytical interest. For example, a casualty log is commonplace among campaign models, where the number of casualties over time is recorded with a reference to the engagement where the casualty was incurred. However, in many cases the specific details about that engagement have to be looked up in another log file, usually called something like the 'engagements' log. This presents a number of problems for the analyst:

1. The data required in order to answer one question is strewn across multiple file locations;

2. Duplication is occurring because the context of the casualty data is meaningless without references to data stored in other log files. However, in order to cross-reference between the log-files, a certain amount of duplication is necessary, thereby wasting storage space.

3. The analyst is constrained to the data provided in these log files. If a new circumstance arises whereby additional data is required, this cannot be easily obtained without modifying the code of the model.

Taking these points into account, it was designed such that log files for output to become a highly customisable component of the GAMOV framework. Instead of attempting to conceive of all the possible use-cases for the output, the log file formats

are now an input that can be declared as part of the model instantiation process. This is achieved by effectively having the model record everything to a single log file, with the bespoke log files being produced by compositing themselves from collated data.

4.7 Chapter Summary

Within this chapter, the concept of the GAMOV framework has been presented as a solution to address many of the issues encountered with model development in DSA. This framework has been presented at both a high-level in terms of its organisation and composition of layers, and at a low-level in terms of sub-system design. Where applicable, the impact that Object-Functional programming would have upon this retrospective design has also been identified, including where it is believed that deviations have occurred.

The next chapter shall now evaluate a range of models developed in Dstl using both Object-Functional via the GAMOV approach and prior methods such as pure Object-Orientation.

5 Case Studies of Extant Models

5.1 Chapter Introduction

Within this chapter, the models identified in chapter 3 shall be evaluated using the proposed evaluation framework and presented as a set of case-studies. These chosen models provide examples that cover both the breadth and depth of the complexity of representation and also the range of implementation approaches that are currently employed, including Object-Functional via the outlined GAMOV approach in chapter 4.

Each of these models, although they proved to be useful in the roles they were designed for, provides useful insights into the improvements needed in terms of model design and implementation; and where an approach such as Object-Functional as understood can potentially lend benefits.

The purpose of this evaluation is to frame the overall models in terms of how they are designed and implemented in order to then analyse what issues this causes to the overall modelling capability.

5.2 Model 1 - Wartime Planning Tool (WPT)

What is the purpose of the model?

The Wartime Planning Tool (WPT) is designed to represent the outcome of a day's worth of combat between two forces.

It represents primarily land-focussed units, complemented by air power.

What categories of analysis capability does this model cover?

Policy Simulation & Planning.

Is the model currently in-use?

Yes (but with caveats). There are multiple versions of the model currently in existence. The base version in Excel and a re-implemented version in Python that is due to be incorporated into the GAMOV framework. The re-implementation of the model revealed some errors that were fixed in the Python version. However, neither version is in active use as a standalone system; although the conceptual ideas of the model are still present as the basis for combat calculations in other campaign models.

What language and/or coding environment is the model written in?

Microsoft Excel Spreadsheet, using cell formulae and underlying VBA.

What coding paradigm is the model stated to follow (or appears to follow upon inspection)?

Primarily procedural, with some supporting modules.

How large is the code-base?

Most functionality is driven via the spreadsheet using in built-in formulae within the cells. There is less than 100 lines of code supporting the model in VBA modules.

Are there any observable anti-patterns?

The model is procedural and does not produce any anti-patterns.

Is the model scalable?

No, due to its implementation within Excel.

What scalability approaches does the model employ (if any)?

N/A

What options does the user have in order to interface with the model?

As the model is Excel based, the users can only input their data through direct input into the spreadsheet itself. The model expert noted that a great deal of care should be taken when inputting data this way. Some of the underlying equations are encoded directly into the same cells as the input data. This means that every calculation of WPT overwrites the input values with the new output values. The analyst has to remember to take a copy of the spreadsheet before processing it, in order to preserve the integrity of the original input.

Typically, what options do users use?

Directly use the spreadsheet as the interface to the model.

Are there any observed or documented reliability issues of note associated with this model?

Because WPT is a spreadsheet model, it is as reliable as the Microsoft Excel installation is on the target machine.

What level of modularity does the software employ?

The WPT, in itself, is a module in the grand scheme of things.

What approaches to testing does the model employ?

Scenario based testing using a dataset that has been externally validated.

How reusable are the components within the system?

Internal modules are not reusable outside of the model. The conceptual ideas of the model are reusable and have been re-implemented in GAMOV.

5.3 Model 2 - Strategic Balance of Investment (StratBOI) Linear Program

What is the purpose of the model?

The StratBOI Linear Program is focussed around understanding the balance of available military capabilities to support anticipated military scenarios/tasks and the associated cost implications. Each military scenario has what is known as an 'Order of Battle' (ORBAT), which outlines the anticipated composition of military capabilities that are required in order to achieve the objectives of the scenario.

The StratBOI study focuses on understanding what balance of investment needs to be made by assessing what can achieved with the available capabilities and assessing whether any shortfalls exist.

The model also takes concurrent obligations into account, whereby the U.K. may be potentially operating in more than one theatre. These obligations may be other military activities, which could be another full military campaign, or supporting the activities of other international bodies e.g. NATO, United Nations etc.

What categories of analysis capability does this model cover?

Policy Simulation, Financial Evaluation, Procurement & Planning

Is the model currently in-use?

Yes

What language and/or coding environment is the model written in?

StratBOI is written in a variety of different technologies in order to deliver the capability of the linear program. This includes the optimiser that is a closed application (essentially a black box), XPressMP, SQL queries and spreadsheet data files.

What coding paradigm is the model stated to follow (or appears to follow upon inspection)?

The model is effectively an end-to-end procedural process of different technologies, where the model is populated via data files, pre-processed to run across the optimiser and then queried in post-process to produce the outputs.

How large is the code-base?

< 1000 lines across the technologies

Are there any observable anti-patterns?

None in terms of the code.

Is the model scalable?

Yes

What scalability approaches does the model employ (if any)?

Uses lazy parallelism to run multiple cases in parallel on HPC.

What options does the user have in order to interface with the model?

Model input is primarily through the data files.

Typically, what options do users use?

Data files.

Are there any observed or documented reliability issues of note associated with this model?

None.

What level of modularity does the software employ?

The model is an end-to-end procedural process of different technologies. Therefore the ‘model’ is procedural outside of some modularity in the individual stages.

What approaches to testing does the model employ?

Primarily uses scenario and externally validated data.

How reusable are the components within the system?

With the exception of the optimiser, which is a commercial product, the other components that make up the StratBOI LP capability are tailored to the model.

5.4 Model 3 - Diplomatic and Military Operations in a Non-war fighting Domain (DIAMOND)

What is the purpose of the model?

DIAMOND is a is high-level, campaign model, intended for the representation of peace-support operations.

The model is primarily focused on the representation of land forces, supported by air assets. Maritime assets can be modelled, but their representations are simplified in order to reflect their supporting role in such campaigns; such as sources of logistics or platforms to deploy land or air assets from or into theatre.

What categories of analysis capability does this model cover?

Policy Simulation, Forecasting (Outcomes of Courses of Action) Planning & Conceptual.

Is the model currently in-use?

No, the model custodian has retired the model due to a number of documented reasons. Principally this is due to the fact that the pre-existing functionality within the model cannot be repurposed quickly to represent new and emerging ideas. Even when the model is able to be reconfigured, the cost benefit is low. A great deal of knowledge within the staff has also been lost about both setting the model up and developing the underlying code-base. The model is also noted as being very difficult to fully understand due to the amount of residual DROMAS functionality that is inherited by all models produced by DROMAS, but not all is used in DIAMOND. This means there is not a clear boundary between the two.

What language and/or coding environment is the model written in?

Visual C++ using Microsoft Foundation Classes. Could is derived from the DROMAS framework, which has also been used to produce other models in Dstl, such as SIMBAT.

What coding paradigm is the model stated to follow (or appears to follow upon inspection)?

Object-Orientation.

How large is the code-base?

Overall, the code-base is estimated to be somewhere in the region of 25,000 lines or more. However, it is not completely clear from inspecting the structure as to how much of this code is DIAMOND and how much of it is DROMAS. From examining a few samples of modules that identify themselves as DIAMOND modules, these are merely less than half a dozen lines in each that are specific to the model.

Are there any observable anti-patterns?

There is believed to be strong coupling in the model (spaghetti code), due to the difficulty in reconfiguring the model to other situations. However, there are no current developers to attest as to whether this is a product of DROMAS or DIAMOND specifically.

Is the model scalable?

The model currently does not scale, but it would be possible to employ a technique such as lazy parallelism to run multiple versions of the model in parallel. Beyond that there is not enough understanding of the model's code structure to draw upon in order to employ anything more sophisticated within cost benefit.

What scalability approaches does the model employ (if any)?

None

What options does the user have in order to interface with the model?

The model is configurable through a GUI and a scenario file.

Typically, what options do users use?

The GUI exclusively. There is no record of setting up the model via the scenario file approach outside of the test scenario that is used to validate the model. The scenario file is flat-file with very specific formatting and contains special wildcard characters.

Whilst the GUI is usable, the custodian does not consider it to be intuitive. This is due to two main reasons. Firstly, data to configure one piece of functionality in the model has to be configured in multiple interfaces; and it is not obvious that all of these interfaces are inherently linked. Secondly, because DIAMOND inherits all of the interface options offered by DROMAS, there are a lot of redundant elements in the GUI. For example, entities within the model are able to fight each other, irrespective of whether you give them any logistics to do so. This compounds the confusion as to how the model works overall.

Are there any observed or documented reliability issues of note associated with this model?

The model is noted to have two major reliability issues. Firstly, the model has a memory leak, which has never been traced down because its source has not proven to be identifiable within in the code-base. Secondly, it is believed that parts of the DROMAS framework are reliant on the operating system for aspects of its configuration. For example, the model can produce different results when ran on different machine configurations, because it is believed that DROMAS is seeding from the system clock. Again, this cannot be confirmed in the code-base as it is not readily apparent where this activity is occurring.

What level of modularity does the software employ?

The code-base is extremely modular to the point that it is over specific. However, as stated earlier, the distinction between DROMAS and DIAMOND makes it difficult to know at what level classes are being derived.

What approaches to testing does the model employ?

Testing is primarily conducted via the test scenario. There are no unit-tests in the DIAMOND code-base or test runner.

How reusable are the components within the system?

Due to the level of inheritance from DROMAS classes, DIAMOND classes are not easily reusable outside of DROMAS based models.

5.5 Model 4 - C3 Oriented Model of Air and Naval Domains (COMAND)

What is the purpose of the model?

COMAND is a high-level campaign model that provides a comprehensive representation of both the maritime and air military domains.

A key characteristic of the model is its representation of the Command, Control and Communications (C3) layer, whereby orders and information updates can be issued to the forces. This gives both a scripted representation of the command and control decision-making that typically occurs within military campaigns and the representation of the interactions of information exchange and dissemination among the units.

The model has a limited representation of ground forces, only including those elements that are either targets for air or maritime assets; or threats to the operation and survivability of the air and maritime assets. This includes elements such as key installations and ground-based weapons platforms. If a more detailed representation of the ground component is required, the data of COMAND is often fed into another model that represents land forces and their operations at a greater resolution.

What categories of analysis capability does this model cover?

Policy Simulation, Forecasting (Outcomes of Courses of Action), Planning & Conceptual.

Is the model currently in-use?

Yes, but with expert oversight.

What language and/or coding environment is the model written in?

Visual C++ using Microsoft Foundation Classes.

What coding paradigm is the model stated to follow (or appears to follow upon inspection)?

Object-Orientation.

How large is the code-base?

Somewhere in the region of 25,000 – 50,000 lines.

Are there any observable anti-patterns?

The model custodian confirmed that the model has a high degree of coupling present (spaghetti code); however, from his experience of the code base, whilst very time-consuming, the code-base is at least concise enough that it is possible to follow the thread of execution to root out bugs.

Is the model scalable?

Yes, at the executable level.

What scalability approaches does the model employ (if any)?

The model is currently configured to run multiple instances in parallel via lazy parallelism. Some small experiments were performed at the time the model was configured to run in parallel, to see whether process or thread level scalability was possible. However, the cost benefit was deemed to be low due to the overhead of understanding the coupling in the code-base.

What options does the user have in order to interface with the model?

COMAND offers a set of GUI forms for data input.

Typically, what options do users use?

In the custodian's experience, once a user becomes familiar with the model, they bypass the GUI forms completely and opt to configure the model using the underlying databases. COMAND uses a combination of a master database and a scenario database for configuration. The master database is applicable to all scenario and the scenario database is specific to the scenario. If users are modifying these, then changes to the master database have to be centrally managed and reviewed.

Are there any observed or documented reliability issues of note associated with this model?

In the custodian's experience, most issues are a result of data input. However, the model is not easily interruptible nor can it be started from a specific point-in time. Therefore, if a crash occurs due to data input many hours into the runtime, it makes it time consuming to replicate and resolve issues.

What level of modularity does the software employ?

COMAND's underlying structures uses a very detailed Object-Oriented hierarchy in order to structure its code components. Whilst the organisation is not overly specific where an object can only represent an instance of capability (e.g. a type-45 destroyer vs. a type-23 frigate) the objects are still specific enough that they can only be used to represent a group of capabilities (e.g. surface vessels). Functionality is both encapsulated within these capabilities, but also a range of generic model capabilities are available as their own object services.

What approaches to testing does the model employ?

Testing is primarily conducted via the test scenarios. There are no unit-tests or test runners present.

How reusable are the components within the system?

In the custodian's experience, it would be faster to reimplement functionality rather than trying to port it, due to the age and coupling of the code, and the reliance on foundation classes. However, the conceptual underpinnings of COMAND are strong and could still be used in such efforts.

5.6 Model 5 - Aerial Delivery Model (ADM)

What is the purpose of the model?

The Aerial Delivery Model (ADM) is a model used for analysis logistics demand across a node and arc network environment.

The ADM examines lift options at the operational level, which focuses on understanding the requirements and challenges of moving logistics or personnel to and through theatre using what is referred to as a 'lift asset'. This would include aircraft such as Helicopters, Chinooks and large carrier planes to name but a few examples.

What categories of analysis capability does this model cover?

Policy, Forecasting (Demand of logistics) & Planning.

Is the model currently in-use?

Model is available, but not being currently used.

What language and/or coding environment is the model written in?

Python

What coding paradigm is the model stated to follow (or appears to follow upon inspection)?

Object-Functional (via the GAMOV framework approach).

How large is the code-base?

~500 lines of code unique to ADM (not including code provided by GAMOV)

Are there any observable anti-patterns?

Not in terms of the model's functionality. As the model follows the approach of the GAMOV framework, it uses the entity hierarchy for representing environment and units and independent mediators for each piece of functionality. However, the ADM is using an outdated version of the GAMOV entity base class. If the model were to be

further developed, this would need to be updated to remove technical debt. Upon reflection of this model, the entity structure in GAMOV, whilst providing a flexible approach to the representation of units can easily fall out of sync with other models if it does not keep pace with changes in the GAMOV framework.

Is the model scalable?

Yes, all functions in the model are first class objects and could be scaled onto HPC. None of the functions are currently programmed to individually scale using parallel processes or threads.

What scalability approaches does the model employ (if any)?

None currently

What options does the user have in order to interface with the model?

Data files only.

Typically, what options do users use?

Data files.

Are there any observed or documented reliability issues of note associated with this model?

None.

What level of modularity does the software employ?

All functionality in the model is its own class.

What approaches to testing does the model employ?

All modules in the model are capable of being tested in isolation through unit-tests and subsystems via integration tests at a desired level.

How reusable are the components within the system?

The ADM is focussed on logistics-based studies, therefore the model as a whole can be reused as the basis for other logistics-based work, because the conceptual ideas

associated with moving logistics across a network could be repurposed from aerial to other ground-based methods. Components such as the route-finding routines to deliver logistics to points of demand could also be reused across other models that are not logistics based.

5.7 Model 6 - Mission Command Model (MCM)

What is the purpose of the model?

The MCM is an experimental model designed to provide an automated command and control core for campaign modelling. The model uses automated mission planning, based upon the mission planner research to produce sets of missions that achieve the overall objective of specific military actions, or courses of action. This would classically be a manual process for the modeller requiring these missions and overall campaign plan to be scripted, which is typically a very time intensive process.

What categories of analysis capability does this model cover?

Policy and Planning (Military Actions and Courses of Action)

Is the model currently in-use?

Model is available, but not being currently used.

What language and/or coding environment is the model written in?

Python

What coding paradigm is the model stated to follow (or appears to follow upon inspection)?

Object-Functional (via the GAMOV framework approach).

How large is the code-base?

~2000 lines of code unique to ADM (not including code provided by GAMOV)

Are there any observable anti-patterns?

Not in terms of the model's functionality.

Is the model scalable?

Yes, all functions in the model are first class objects and could be scaled onto HPC. None of the functions are currently programmed to individually scale using parallel processes or threads.

What scalability approaches does the model employ (if any)?

None currently

What options does the user have in order to interface with the model?

Data files only.

Typically, what options do users use?

Data Files

Are there any observed or documented reliability issues of note associated with this model?

None in terms of stability; however, the model has experienced issues with respect to memory management associated primarily with the entity structure. Due to the optimiser testing many different outcomes of plan, nested entity structures are being copied lots of times and not being destroyed consistently when a plan is no longer in scope.

What level of modularity does the software employ?

The model follows the approach of the GAMOV framework; thus, all functionality is within its own functional classes.

What approaches to testing does the model employ?

All modules in the model are capable of being tested in isolation through unit-tests and subsystems via integration tests at a desired level.

How reusable are the components within the system?

The MCM is focussed on delivering a core component that could be used for the automation of command and control in other models. From that perspective, the MCM is reusable in any model with a command and control requirement. However, the MCM is currently tailored towards planning missions that occur on a node-arc environment. Thus, the component would have to be extended if it were to be reused in other modelling environments that operate on free movement or grid-squares.

6 Analysis

6.1 Chapter Overview

Within this chapter the information collated from evaluating the models shall now be analysed in order to further understand the implications of their individual implementations.

6.2 Analysis of the Wartime Planning Tool (WPT)

The WPT is an example of what historically made up a large portion of the model development activities within Dstl, which is using spreadsheet packages (principally Microsoft Excel) supported with built in formulae, macros and Visual Basic for Applications (VBA) code. In experience, the main driving force behind using this approach (and more importantly, continuing to use it) is with respect to deployment of the software. The majority of the customers that Dstl are supplying modelling software to are running basic computing setups with strict security models. Therefore, it is unlikely that these setups will have access to a programming language such as Python or Java, unless it is bundled as part of the system setup. Even then, being able to customise the setup of these languages to a specific model is also unlikely. As a result, for a long time there has been an inherent convenience, in terms of deployment of models to customers through using VBA with Microsoft Excel. However, with the advent of languages such as JavaScript, which is effectively available to anyone running a web browser, there are now alternatives to this approach. The continued usage of this approach for modelling actively presents a number of problems, which shall be highlighted in the next few subsections.

6.2.1 Poor Implementation

With VBA being readily available, with only Microsoft Excel being a pre-requisite, the developer base for VBA in Dstl is extremely broad but technically shallow in terms of software design experience. In experience, this has produced models displaying a variety of issues, which can be observed in the WPT implementation. For example, some of the underlying equations of WPT are encoded directly into the same cells as the input data. This means that every calculation of WPT overwrites the input values with the new output values. The analyst is then responsible in remembering to take a

copy of the spreadsheet before processing it, in order to preserve the integrity of the original input.

In light of the problems faced with spreadsheet modelling quality, considerable effort has been made within Dstl in order to provide additional training and support for these developers, including the development of dedicated software expertise in-house to provide advice and support. Wider government has also supplemented these efforts with the production of guidance, most notably the Aqua Book (HM Treasury, 2015) for quality assurance in the production of models and products from the analytical process.

6.2.2 Multiple pass process

WPT is an example of a spreadsheet model that operates with a feedback loop, where the outputs of the one calculation are required as input to the next. Depending on the model in question, this requires a lot of manual data input and pre-processing activities to be undertaken for each calculation in the loop.

6.2.3 Reduced capability to test code

Whilst there are a number of open-source projects such as Rubber-Duck (Rubber-Duck, 2014) that allows for approaches such as unit-testing and configuration change control of VBA code; VBA on the whole is very difficult to test, as it does not easily allow for modern development tool integration. As a result, verification can be difficult and there are a number of instances where verification errors have persisted, unnoticed within models for quite some time. This falls in line with the findings presented by Roy, Hermans and Van Deursen (2017), which showed that most users of spreadsheets in their studies tested them with scenarios rather than automated methods and thus did not catch severe instances of errors in their implementation. The WPT case study also illustrated an example of this through the re-implementation of the model from its extant spreadsheet version into a Python function in the GAMOV framework. The re-implementation of the model uncovered a number of logical errors in the code that had persisted for many years, which would never have been uncovered otherwise.

6.2.4 Performance

Many of these spreadsheet models have been progressively adapted over many years and are pushed beyond the means that Excel was designed to handle. As a result, the performance of such tools can become very poor. The ADM case study illustrated an example this. The re-implementation of the original ADM spreadsheet into the GAMOV framework produced a model that overall performed better and was more in line with reality. When the code was examined further, the GAMOV team deduced that the when lots of frequent logistics drops were being made, the random number generator in Excel was effectively breaking down and was unable to cope with so many frequent call-backs. Analysis undertaken by McCullough & Wilson (2005) concluded that older versions of Excel prior to 2010 displayed many issues with RNG in terms of accuracy and performance that made it unsuitable for this kind of work, which would have been the timeframe that the original ADM was conceived. The GAMOV version of the model did not experience any of these problems, due in part to being ran in a language suited for the purpose. Whilst it is possible that innovations in later versions of Excel may not encounter these problems, by being part of the GAMOV framework, the modeller now has the option of using different distributions and their testing accuracy and performance; rather than being constrained to what is shipped with Excel.

6.2.5 How could Object-Functional help WPT?

For the most part, spreadsheet models within Dstl represent what would be considered either single functions or a sub-system that would be found within a larger campaign model. In fact, there are many cases where spreadsheet models provide direct inputs to the campaign level models. Therefore, by incorporating these spreadsheets into any campaign model irrespective of implementation would bring benefit in this regard. Firstly, they could feed directly into the model they are supporting and secondly, they could be queued up to be re-ran multiple times removing the mandraulic process of copying output from one spreadsheet run to be input for the next. However, Object-Functional would provide the benefit of making this a de-coupled component of a wider model, meaning that conceptual ideas like WPT and ADM are a service of the wider system rather than being encapsulated as a method of one of the model's entities.

However, Object-Functional would provide benefit with respect to testing these spreadsheets, because each spreadsheet would become a functional class, or a mediator under the GAMOV approach. This means that they become callable (RESTful) services with a defined interface that can be tested independently with unit-test and as part of the wider system of systems with integration tests. Additionally, services that these spreadsheet models depend upon, such as random number generators, would also be functional classes or mediators. This means that different distributions and implementations of random number generation can be plugged and unplugged from these models depending on either the changing requirements of the model or the evolution of the science behind pseudo random number generation.

6.3 Analysis of the StratBOI Linear Program

The StratBOI toolset is composed of a number of tools in order to answer the questions of the overall programme. The tool that specifically optimises the balance of forces to objectives is the StratBOI Linear Program (StratBOI LP), which is what was examined via the case study.

6.3.1 Using proprietary software for models

Whilst the StratBOI LP is not a campaign model, it provides an example of a common implementation method that is sometimes adopted for the development of campaign models within the organisation. That is the incorporation of licensed, third party (off-the-shelf) software tools working in cooperation with bespoke code developed in-house. In in the case of StratBOI LP, this is an optimisation routine, but in others it may be the underpinning software components of a larger modelling representation. This presents a number of key benefits:

- Reduced Maintenance - Using proprietary tools can reduce the amount of internal maintenance of the overall software. Whilst the software will still need to be validated for our purposes, we are not wholly responsible for verification in terms of how the product functions.

- Professional Support - There is the assurance that problems in the software will be patched and critical updates will be received promptly for licence holders. A common problem with internal software development is the incurring of

technical debt, which is where a problem in the software has to be left unrepaired or worked around in the interests of time and money. Having external professional support helps to reduce that technical debt with respect to the proprietary components.

However, there are a number of challenges with using licenced software for the purposes of our modelling. These shall now be outlined in the next few subsections.

6.3.1.1 Black Boxes

Given that these software products are proprietary, there have been instances where some of the underlying code is deliberately obfuscated due to the terms of the licensing agreement. Whilst the software may have an API that will explain the inputs and outputs to functions and the overall tolerances of the software, in practice, it is often very useful for our purposes to see and understand the algorithms in order to validate the representation. This goes back to the common issues associated with models raised by Salt (2008), where black boxes impede our ability to provide comprehensive assurance, because there are aspects of the system, we are relying upon that we do not understand.

6.3.1.2 Complication to inputs

In some cases, the input data for a scenario has to be modified in order to account for the behaviour of the software, either because it cannot be changed or it is intrinsic to the functioning of the software. For example, in the case of StratBOI, there are aspects of the problem that are inherently non-linear that have to be modified so that they can be processed by the linear optimisation routines used within the proprietary component. This could be solved by migrating to a non-linear optimisation approach such as beam search or genetic algorithms, however, the culture of data preparation, input and output of StratBOI would also have to evolve as they are somewhat slaved to the dependencies of the original implementation.

6.3.1.3 Cost

Using these tools often requires payment for use of a license. Given the overarching need to reduce costs, the usage of these products has to be balanced against a study factors in order to at least justify the investment.

6.3.1.4 Implementation Specificity

The use of proprietary software can drive the design of any bespoke elements that are interacting with it. Therefore, if the proprietary software became unavailable through business closure or suspension of support etc. there are potential implications to the bespoke code written in-house. This is also true for software that relies upon the workings of the operating system in any way. For example, built-in random number generators, which could change as part of an operating system upgrade. This also came out in the DIAMOND case study, where it at least appears the DROMAS framework is dependent on such configuration in order to function.

6.3.2 Using proprietary software for campaign models

It is not unheard of within Dstl for third-party licensed modelling software to be used to enable the entire implementation of a campaign model, using software such a Simul8. However, the majority of the work that uses proprietary products in this fashion is usually looking at subsets of the campaign in more detail, such as logistical problems, casualty evacuation etc. where the outcome is an optimisation of a system; as in the case of the StratBOI LP.

Campaign models are by their very nature far larger than many of these proprietary products were designed to handle. They also possess unique characteristics with respect to their representations that are not necessarily going to be ‘out of the box’ functions for many products. This comes down to a few key characteristics, which shall be outlined in the next few subsections.

Compared to the majority of other fields that are using modelling and simulation, defence campaign models represent an antagonist (i.e. the enemy forces) that are actively working to undermine the system in order for their plans to succeed. In our experience, whilst most third-party modelling software can represent disruptions to a system; it is difficult to represent an adversary force that is actively planning in response to the actions being made by its opposition. This fundamentally comes down to the maturity of ideas in representing command and control within models as presented by Moffat (2011). However, this is one of the reasons why we opt to build models that represent these networks of decision-making using bespoke tools, so that we can tailor the representation more towards our needs rather than effectively kludging an idea using a pre-built tool.

Coupled with having multiple sides being represented; those sides are not necessarily representing just one force. Quite often a single side can be made up of multiple forces from either different nations or sub-factions of a single nation. This creates a great degree of interplay within a single side, which can be difficult to represent outside of bespoke code due to inbuilt assumptions about how groups of actors work within a simulation.

Quite often the forces at play within the models (even if they are represented as being on the same side) are driven by other personal objectives that will impact their behaviour within the simulation. These may be policy decisions or social factors to name but a few; however, these are often difficult to represent outside of bespoke tools again due to pre-built assumptions.

6.3.3 How can Object-Functional help StratBOI?

Whilst StratBOI does not suffer from many of the issues encountered by models such as WPT, DIMAOND and COMAND as will be discussed later; the model is heavily dependent on a black box component with respect to its optimisation routine. Due to the proprietary nature of this component, the separation of data and functionality brought about by Object-Functional would not provide any additional benefit to StratBOI because it already exists in that regard. However, the StratBOI optimisation routine could become a mediator in the GAMOV framework in order to increase its utilisation across other problem spaces outside of StratBOI. Additionally, as with WPT, this would allow additional testing harnesses to be attached to the optimiser to better understand its functional behaviour. Whilst the optimiser would still be a black-box conceptually, the ability to test at the point of contact rather than relying on scenario level testing could provide additional assurance.

An observation from the evaluation that was recorded, but did not have an explicit question in order to capture it, was that StratBOI is an example of a model, whose outputs are a result of what could be considered emergence. The implications of emergent phenomena shall be discussed later in more detail within section [7.4.2] when the future research work is outlined. However, at this stage it is important to understand that the presence of emergent phenomena presents an opportunity to learn more about what is being modelled through the simulation (Georgiou 2007). They offer significant insight into the functioning of the systems, such as what

circumstances are required to produce certain events (or not produce an event). Campaign models such as DIAMOND and COMAND will also produce this kind of emergence due to the heavy interplay between the many systems represented in a campaign model.

Whilst the solution produced by StratBOI can be validated by SME's to ensure that it is sensible, the ability to study the interplay between the variables of the model could offer greater insight into why one solution was favoured over another during the optimisation process. However, there is no automated tool or method in existence that allows an analyst to engage with emergent behaviours. This highlights a significant challenge associated with emergence in general that has yet to be addressed (Sterman, 2000): Provision of an automated means by which to engage with emergence within a constructive simulation. Whilst this is not the focus of this thesis, the implications of emergence are an important consideration and it is believed that Object-Functional could potentially aid in grappling with this complexity. By having all functional elements of the system existing as functional classes and the data modified through interfaces; the organisation of components inherent with Object-Functional is believed to provide significant granularity to test flows of information within the system. As emergence is concerned with understanding when and why something has occurred, having the granularity in the system to pinpoint this down would be key.

6.4 Analysis of the DIAMOND Campaign Model

The DIAMOND case-study unfortunately paints a much bleaker picture, as this is a model that has been retired due to the fact that its implementation had become so restrictive that it was no longer fit-for-purpose. Building representations within DIAMOND are no longer cost effective because of this, but also the conceptual underpinnings of the model have greatly evolved and could not be changed. The modeller is not only forced to repurpose functionality that is deficient in terms of its implementation but also in terms of what it is conceptually representing.

6.4.1 Usage of DROMAS

Examining the model from an implementation perspective, DIAMOND is one of a number of models that is based upon a framework known as DROMAS, which was a precursor attempt to realise a reusable component framework similar to the GAMOV.

However, DROMAS was rooted in realising this through Object-Orientation and did not at explicitly consider it in the terms of the Functional paradigm, albeit it was striving for reusability in pure Object-Orientation.

DROMAS intended to provide a component architecture whereby models and their GUI could be quickly implemented, but as a result constrained the range of potential solutions. However, DROMAS lacks the flexibility that would be expected of a reusable framework, because the policy for how the components should be assembled is overly restrictive. The examination undertaken of the code-base, whilst very high level did suggest this might be the case. Firstly, it was very difficult to identify where the boundary between DROMAS and DIAMOND existed. Secondly those classes that identified themselves as being DIMAOND by name, merely contained code that were parametrisations of DROMAS functionality. This means that DIAMOND in reality is just parameterisation of DROMAS and does not have the option to add capability to the framework when compared to something like GAMOV. As a result, all models produced from DROMAS are merely slight variations on a theme with extremely similar looking GUI's and performance. To compound this issue further, the API for DROMAS is extremely difficult to interpret for new developers, due to documentation quality and a lack of knowledge retention. Therefore, DIAMOND and other models produced from the DROMAS framework have been difficult to maintain and adapt, with instances of these models, including DIAMOND been retired from service.

6.4.2 Code and Concept are coupled

The DIAMOND model was originally implemented and validated against the MODs understanding of the requirements for conducting a peace support operation, based upon the events surrounding Bosnia. However, with the advent of conflicts such as the Iraq and the Afghanistan wars in the 2000's, our understanding and policy regarding peace support operations has fundamentally changed. Afghanistan and Iraq are examples where peace support has evolved into a more long-term and enduring military task, with more involved activities, such as repairing the infrastructure of the nation and building trust with the populace. Enemy tactics have also changed, with more focus on terrorism and insurgent based tactics, which requires a different set of representations within the models. In the case of DIAMOND, the original representations are believed to be hardwired into the code as

functionality, rather than through data. As a result, inputting a new scenario into the DIAMOND model is extremely time-consuming. This is because the modeller is forcing the representation of the scenario into the pre-existing functionality defined by DROMAS for another purpose. Thus, the modeller is working towards realising something that is close enough to being considered reality rather than grappling with the reality itself.

6.4.3 Coupled GUI

DIAMOND is also an example of a model that was heavily coupled to its GUI. The GUI component of the DROMAS framework underpins the construction of models from the framework. As a result, nearly all input and output from the model occurs through the GUI, with a heavy reliance on forms. This makes data input to the model quite long-winded because of the number of visual elements that needed to be interacted with to simply add something small, such as a new unit. Additionally, the underpinning data files of models based on DROMAS were not easily human-readable. As a result, the GUI becomes the only viable option for model interaction.

However, outside of the usability issues this causes, the fact that every DROMAS model GUI is the same creates an operational issue for the modellers. Because every GUI option provided by DROMAS is inherited and represented in the final product there is redundant functionality in the DIAMOND model GUI. As illustrated in the case-study, DIAMOND has the GUI option to give entities the ammunition to fight, but they will fight and inflict damage upon one another whether they have ammunition or not. As knowledge retention for the model has already proven difficult, these compounds the understanding for new modellers, as the scope of the model capability is very unclear.

6.4.4 How can Object-Functional help DIAMOND?

Unfortunately, as DIAMOND appears to be a parametrisation of DROMAS it is not a model that can stand to benefit from an Object-Functional approach directly, because it is by its very nature a reconfiguration of a framework, rather than a model being a product of a framework. DROMAS itself would have to be changed to conform to Object-Functional.

The Object-Functional approach does however discourage the practices brought about the design of DROMAS style framework. One could even go as far to say that Object-Functional is the anthesis of this approach. Because Object-Functional strives to make all methods functionally independent via functional classes, they become off-the-shelf services, with their configuration left to the modeller to define. DROMAS unfortunately defines this configuration for the modeller, thus reuse is diminished unless the component is being used in a context that is the same or very similar. In this way Object-Functional not only allows you to define the model, it also allows you to define the framework of that model.

The GAMOV approach to Object-Functional does provide some constraint on the model framework, but only with respect to how the data and functionality interface works. This is a necessary constraint to preserve this interaction and ensure consistent behaviour, but does not preclude the user from plugging components together in whatever fashion they desire, in order to build a model. This could potentially allow modellers to build something that is invalid or not fit-for-purpose, but that is not the responsibility of GAMOV; the onus is on the modeller to validate their GAMOV solution. However, the flexibility afforded by this setup would allow the user to rectify this. DROMAS seems to have taken the approach, explicitly or implicitly of not allowing the modeller to make any mistakes in most areas, but that has come at great expense to its flexibility and ultimately led to the DIAMOND model being discontinued.

6.5 Analysis of the COMAND campaign model

The COMAND campaign model paints a much brighter picture when compared to the DIAMOND campaign model, because it is still experiencing continued use. All elements of the COMAND model are written in Visual C++, using Microsoft Foundation Classes (MFC). This includes the business logic, which consists of highly specified Object hierarchies and the GUI. This gives the tool a familiar look and feel to the overall Windows environment that it runs on. However, COMAND stands as a prime example of the issues and anti-patterns brought about by the various interpretations one can make of the Object-Orientation paradigm, which shall be explained in the next subsection.

6.5.1 Anti-Patterns present in COMAND

Upon examination of the code, COMAND's underlying structures uses a very detailed Object-Oriented hierarchy in order to structure its code components. Whilst the organisation is not overly specific where an object can only represent an instance of capability (e.g. a type-45 destroyer vs. a type-23 frigate) the objects are still specific enough that they can only be used to represent a group of capabilities (e.g. surface vessels). COMAND's implementation highlights the problem associated with tight code coupling commonly associated with Object-Orientation. Functionality in the COMAND model is owned by the entities themselves as the per the concept of encapsulation.

The lack of both interdependence and interfacing between the model's code elements means that the functionality is potentially sensitive to small changes and the model custodian confirmed that this often produces significant side-effects in other areas of the code. COMAND has now in fact reached a point where it cannot be further adapted due to the production of these side effects at practical cost. As a result, COMAND is becoming progressively more limited in its ability to represent scenarios, without incurring significant overhead in the model setup time.

COMANDs code has elements of the 'spaghetti' anti-pattern due to the tight coupling between some of these components. However, there is no obvious 'blob' pattern in the code to further compound the maintenance. As discussed in the literature by Abbes, Khomh and Gueheneuc (2011), the existence of spaghetti code alone can be managed provided it is not compounded by a blob. This falls in line with the

experiences of the model custodian who has experienced and observed that most COMAND developers can navigate the code to root out problems, albeit to greater time and cost if the code were not organised this way. However, as the COMAND code is now so sensitive to change, this also suggests the presence of a ‘lava flow’ anti-pattern (Webster, 1995) in potentially multiple places. This means that the coupling has produced functional dependencies that if changed would stop other parts of the model working entirely.

COMAND also suffers from the fact that data, representing behaviours has been encoded into the functionality of the model, which cannot be easily modified through other forms of data entry. The custodian suggested that this likely occurred as a result of time pressures in the development history of the model, however, this is now technical debt that cannot be easily addressed. The presence of this data can put significant limitations on the representation of certain elements (in some cases, prohibiting a representation of entire aspects of a scenario) and extends the model setup time. Often, the analyst is required to script additional tasks to force the components to behave in the correct manner. For example, certain elements of military policy (that were correct back when the model was originally constructed) have been encoded into the model, which may force units to commit or retreat from certain tasks due to the emergence of certain conditions. The analyst may either have to script a task multiple times, or manually remove the conditions that force a unit to commit or retreat in order to ensure that a scripted task is completed in expected timeframes of the scenario.

Given the progressive limitations upon the adaptability of COMAND’s code modules, the model is often unable to represent novel elements that are starting to emerge in the military domain; for example, Cyber. Whilst it may be possible to script the effects of something such as a Cyber-attack, representing point of origin and transmission of the attack through a network of capabilities is not possible. Whilst information can be passed between the forces, the resultant effect on their behaviour is very limited. For example, it is possible to interpret from the model what an individual unit knows about its current surroundings and the knowledge it possesses about where other units are positioned. However, units for the most part cannot react to this information to either change its course of action or influence other units’ behaviours. To a certain extent this comes down to a lack of understanding of ‘perception’ to which significant research effort is being devoted across the lab. However, the main limitation for the

model as a piece of software is the inability to represent the impact of dynamic command and control network.

6.5.2 Data Management

Data is primarily fed into the model using a Microsoft SQL Server database; however, some data can be directly fed into the model via its GUI. COMAND data is typically structured into two databases:

- Authorised Database – Contains scenario-independent data that is used by all scenarios represented within COMAND. This primarily includes performance data of the military capabilities, such as movement speeds, firing rates, fuel capacities etc. Changes made to the authorised database have the potential to affect all pre-existing representations made in the model; therefore, changes are strictly controlled.

- Scenario Database – This database contains the data values pertaining to the specific scenario being represented. Every military scenario that is represented within COMAND has one of these databases. Common data elements that can be found within this database include the forces structures, key locations, target and other elements such as weather.

On average the model custodian estimated that it takes, at a minimum, 2-3 months for a new scenario to be built and tested from scratch. However, this is a rough estimate, because this is heavily dependent on the size of the representation and or the presence of a similar scenario that could be adapted into a new one. The separation of data using the approach of two databases is effective in terms of control changes in the model and the individual scenarios; and reflects the approach that GAMOV takes in how it manages its entities. The authorised database in GAMOV would be the GAMOV entity template and the scenario database would be the specific model entity. However, the configuration management of these files can be an overhead. This situation has greatly improved with the decision to adopt a Microsoft SQL database solution in the models later life, but previously COMAND used to use Microsoft Access databases to store its data. The problem under Access is that the combination of the authorised and scenario databases only produced one representation of the scenario. If another variation of the scenario was required, a new copy of the scenario database including these changes had to be produced. This

could make the tracking of changes difficult because if a fundamental change is required to the scenario, then this has to be replicated and kept in sync across all scenario files.

Compared to DIAMOND, COMAND's data is much easier to access and is presented in a human-readable format; therefore, the analyst has the option to use either a database GUI or the data files directly. COMAND therefore has much less coupling compared to DIAMOND with respect to its GUIs; however, some coupling between individual data elements and the model GUI still exist. This means that changes to the underlying business logic of the model have to be reflected through on the model GUI. COMAND also has a visual representation of the scenario, including the geography, the location and status of the assets. Additionally, data can be input into the databases through this representation is and through a number of forms for data entry. However, in practice, it has been observed that the analysts seldom use this approach, preferring direct manipulation of the databases themselves.

Lastly, in order to visualize the scenario on the GUI, COMAND requires information on the geography. This is supplied to the model in the form of a map file, which consists of a sequence of 0's, 1's and 2's values (0 representing water, 1 representing land and 2 representing coastal divide). This means that for the most part, an analyst is required to draw the terrain by hand, with the assistance of some input tools. This in particular can be a cumbersome and time-consuming aspect of creating a new COMAND scenario. Not only is the analyst required to interpret geographical data by hand and eye, they also need to consider aspects such as the curvature of the earth implicitly within their representation.

6.5.3 How can Object-Functional help COMAND?

The key benefit that Object-Functional could bring to a model such as COMAND is the elimination of the anti-patterns present within the code-base. Given the coupling present, the conceptual ideas within COMAND would need to be re-implemented into an Object-Functional structure, rather than reorganising the current code-base. Because all functionality is broken into functional classes in Object-Functional, as opposed to encapsulation as they are currently in COMAND, then the coupling would become much looser but explicit as per the goal outlined by Ottinger and Langr (2011).

The analysis of the ADM and MCM models will help to illustrate this further later in the analysis. This would allow COMAND like functionality to exist as off-the-shelf (RESTful) and be reused across many different entity types, as opposed to functionality being shared by the owning entity or potentially duplicated for a subtly different purpose.

6.6 Analysis of the ADM

The original development of the ADM was designed as a test in order to demonstrate that GAMOV could rapidly realise small-scale simulation that would normally be produced using either a VBA spreadsheet model, or a COTS based solution such as Simul8. The objective of the test was to re-implement the pre-existing version of ADM that was written in VBA, using the GAMOV framework.

The ADM examines lift options at the operational level, which focuses on understanding the requirements and challenges of moving logistics or personnel to and through theatre using what is referred to as a 'lift asset'. A lift asset would be described as some form of aircraft with both a capacity to carry logistics, but also the mechanisms required to deploy them through some form of an air-based drop. This would include aircraft such as Helicopters, Chinooks and large carrier planes to name but a few examples.

Specifically, for this model, it is representing the movement of logistics from a main distribution point to delivery points where a demand for logistics has been made. The output of the model is the understanding of how effective the lift assets available within theatre can provision logistics to those points of demand. The ADM was in the process of being re-implemented from its VBA implementation into Simul8. The purpose of GAMOV representation of the model was to demonstrate that we could achieve the same task potentially reducing our reliance on licenced software.

The ADM implementation in GAMOV uses all of the core subsystems of the GAMOV model and made extensive use of the entity and mediator structures. For the representation, we used a central pod and a number of delivery points, all of which were represented by an entity structure. The transport vehicles were also represented using entities, as well as the logistic deployment systems and the air dropped options.

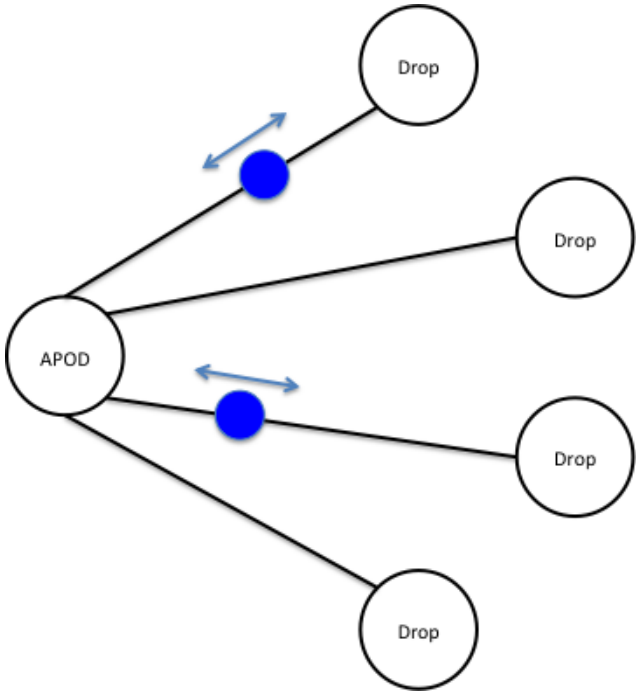

Figure [8] – ADM Layout

6.6.1 What does Object-Functional bring to the ADM

6.6.1.1 Detailed Entity Structures

The original ADM design only took into consideration the movement and deployment of logistics to points at a very high-level. It did not take into account some of the wider issues associated with logistics, such as:

- Deployment mechanism - In other words, the mechanism that is fixed within the logistics carrier used to deploy the logistics.
- Pallets - The containers holding the specific type of logistic. Pallets are typically reused but may sustain damage.
    - Implicit with this is the pallet recovery and repair feedback loop, where some pallets may be taken out of action for a short duration for servicing and thus would not be able to provision logistics during this time.

The GAMOV entity structure is used to incorporate some of these wider issues within the representation. This was achievable due to the fact that entities can hold other entities, which resulted in the following structure for ADM:

- Carrier Entity - The Entity, which moves logistics to and from the APOD to a delivery point. These entities hold:
- Deployment mechanism Entity - The specific system within the Carrier that holds the logistic pallets in place and deploys them upon arrival to the delivery point.
- Pallet Entity - These are the actual pallets that hold logistics of a particular type. These Pallet entities are then held by an instance of a Deployment Mechanism Entity, which in turn are held by an instance of the Carrier Entity. As pallets are deployed, they are transferred from being held by the deployment mechanism to being held by the entity that represents the delivery point.

This provides a richer and more comprehensive picture of the different systems involved at the entity level in terms of what occurs within an ADM type problem. Also, each of the entities has the potential to be offloaded to other entities, which could provide a more flexible system where entities are passed to other entities physically rather than just their data.

Lastly, by putting the ADM into the GAMOV framework, there is now the potential to incorporate it as part of a wider constructive simulation (as was the case with the WPT model). The ADM view makes up only part of what is considered to be the wider intra-theatre logistics problem to be analysed.

6.6.2 What does Object-Functional take away from ADM?

Nothing with respect to Object-Functional itself added overhead to this model. However, the GAMOV implementation via the entity structure showed some conceptual overhead. Upon examining the codebase, the ADM was discovered to be a using a significantly outdated version of the base template for GAMOV entities. The means that ADM is now out of sync and its entities are now suitably different to other models. This presents a conceptual overhead that will need to be managed in order to make the entity solution in GAMOV effective.

6.7 Analysis of the MCM

The development of MCM was intended to realise the aspiration of a dynamic command and control system within a campaign model (Moffat, 2011). The GAMOV approach was used to produce this model so that it could be reusable command and control core for the basis of future models (Glover, Collander-Brown & Taylor, 2017).

The MCM model was built upon the Mission Planner research, (Lucek & Collander-Brown, 2014) which aims to alleviate what has been up to this point mandraulic process, by automating the generation of courses of actions. In other words, it generates the sets of missions needed in order to complete the individual military activities of the campaign plan. This is normally something that would have to be manually scripted by the analyst, with subject matter expertise from the military. However due to the various implementation issues in models such as DIAMOND and COMAND, the process of scripting these missions are cumbersome and time consuming. This is co

The mission planner is a genetic program combined with simulated annealing that is designed to optimise a sensible plan that will achieve the outcomes of the campaign. Like all optimisation algorithms of this nature, the mission planner is susceptible to the limitations in human knowledge about the situation being modelled. If the correct balance of what is permissible and not permissible is not correctly communicated to the planner, it will optimise a solution that is not representative of real-life. However, the capabilities for the planner to automate the generation of plans and the savings in time it will potentially provide makes it a key piece of research.

The Mission Planner would effectively complete a trio of previously developed algorithms that solved other aspects of campaign planning. These are the rapid planner, which was already incorporated into GAMOV framework. Alongside that is the Deliberate Planner, which is seeking to replicate what is referred to as the deliberate, planning process. In real life this would be where a commander(s), prior to the start of a campaign, would plan missions and assign forces in order to meet the overall objectives of said campaign. However, the deliberate planner only worked in terms of assigning the correct force mixes to pre-scripted objectives supplied by the analyst. The sequence of missions still required a lot of heavy scripting in order to be represented within the model. This is traditionally a very time and resource intensive

process as part of the model setup. It's also important to remember that this level of effort only results in one course of action for the campaign. If the model reveals that there are faults in the course of action, or the course of action is changed through military or SME input, then a whole new representation may need to be scripted again in the model.

6.7.1 What does Object-Functional bring to the MCM

6.7.1.1 Rapid incorporation of a new idea

Exploitation of the GAMOV approach to Object-Functional was arguably the only viable route (outside of an entirely new model development project) to exploit the mission planner research. Models such as COMAND and DIAMOND would have been prime candidates to exploit the benefit brought about by mission planner, due to the length of setup time required in order to script their plans. However, due to the amount of coupling found within these models, incorporation of the mission planner code into the existing code-bases was completely impractical, if not impossible within practical cost. The modular nature afforded by functional classes in the Object-Functional paradigm was believed to provide the means to realise the mission planner algorithms as a mediator that can be reused by other models.

The implementation of MCM model is still very much a work in progress and culminates the largest and most complex model built using the GAMOV approach to date (Glover, Collander-Brown & Taylor, 2017).

6.7.2 What does Object-Functional complicate in MCM?

6.7.2.1 Layered Reuse

Assessing the reusability potential of the MCM, highlights a need to differentiate between mediators more than just their functional independence. Throughout the development of MCM, it was acknowledged that there needed to be more of a distinction between mediators in terms of the GAMOV layer diagram. Up to this point, GAMOV mediators had only been conceptualised and described as independent components. Whilst this is true terms of looking at them as software, this did not hold true in terms of the model. MCM highlights that there are going to be special cases where only certain types of mediator can be used in certain types of model and thus, they should be classed differently to those mediators that are truly independent. This typically falls in line with Fielding's (2000) requirement to have a layered system

within REST, where at least conceptually certain components should not interact. The mission planner mediator is actually made up of a number of subsystems in order to form what would be described as a mediator; however, it could not necessarily be used in every type of model being built from GAMOV. The same can also be said for the mediators that handle movement across node and arc networks. If a model uses another form of environmental representation, which for example may be coordinate based, then it will have to use a different type of mediator to handle movement. Even though both ADM and MCM have been using node and arcs to-date, this requirement had not been apparent until MCM development, because we had only been building node and arc-based models. This is more of a distinction in terms of aiding in communicating the concepts to people outside of the team rather than in aid of the developers understanding. However, this distinction is important, so that if someone less experienced is designing a model that there is some clarity to mitigate instances where they try to combine mediators that would not work for their class of model.

6.8 Summary of findings regarding Object-Functional

Following the qualitative analysis of the models used in this study, this section shall now collate all of knowledge gained with respect to Object-Functional. The purpose of this is to frame all the findings in preparation for the conclusions that shall be presented within Chapter 7. Many of these insights were discussed earlier within this thesis; however, this section shall pull all of that learning together into one place.

6.8.1 Status of the Object-Functional Paradigm

The Object-Functional paradigm appears both modern and powerful. This was confirmed by the literature by Lau 2015 and Erikson 2012 and an external assessment of the GAMOV concept undertaken by Boakes and Hanley (2017). However, the understanding of this paradigm is still very much evolving. The amount of academic literature explicitly referencing the term Object-Functional is still low, with most of the understanding currently existing within sources such as blogs, with Lau (2015). The paradigm also appears to have other definitions in common use within the literature, such as ‘functional classes’ as expressed within Kontio, Mäyrä and RÖnkkÖ (2007) and It is possible that there are other works that are following the paradigm, but have not necessarily acknowledged the fact or are using a bespoke descriptor (as in the case with GAMOV acknowledging the paradigm late on in its

development). More formal publication is needed under the name of this paradigm to help bridge this gap in nomenclature and understanding.

As highlighted by Sousa and Ferreira (2012), there is a lack of formal design patterns for the Object-Functional paradigm. This research presents the makings of a pattern, from a high-level perspective, in terms of code organisation through the GAMOV framework. This can be seen chapter 4, which highlights the Entity-Mediator interactions, which explains how functions are scheduled and processed. However, a low-level implementation pattern would still be required in order to effectively communicate the approach to developers who are using it in order to protect the underlying architecture of software projects. For example, the GAMOV entity-mediator interaction, in terms of passing functions and state between the two, could be achieved in more than one way through code. The associated implications of using each approach needs to be further understood and documented in the context of other projects as they present their findings.

Whilst there is a lack of formal design patterns for Object-Functional, the understanding of the intent behind the organisation of the code can be seen by extrapolating ideas from pre-existing protocols, such REST, which was communicated by Boakes and Toomey (2012) and tuple space message passing models (Ahuja, Gelernter & Carriero, 1986). Whilst there is no evidence presented in this thesis to suggest that Object-Functional, REST and tuple space languages are directly influencing one another, there is at least appears to be an association of ideas between the fields that these paradigms are concerned with. This was also an implication of the conclusion of Sousa and Ferreira (2012), where extant patterns can easily be extended, extrapolated or combined to produce new patterns such as Object-Functional.

6.8.2 Benefits of Object Functional

From the examination of the literature and the application to defence models thus far, the benefits of Object-Functional appear to be:

- Reduced Coupling – The Object-Functional organisation of code provides a loose but explicit coupling, which is the goal to removing the tight coupling associated with Object-Orientation as highlighted by Ottinger and Langr (2011). This helps in controlling the spread of side effects as a result of changes to the

code base. This can also be expressed as the removal of anti-patterns as hypothesised by Sousa and Ferreira (2012) to which both ADM and MCM do not have any present.

  - Object-Functional is not about discarding any of the concepts found within its constituent paradigms. I would argue that all of the Object-Oriented and functional constructs that were critiqued are still useful and serve a purpose within Object-Functional. It is simply a case of finding what is the best application for these constructs to yield their benefits for a particular problem. However, from the experiences on developing the GAMOV framework and building models from the framework, I would commend Object-Functional as a powerful framework for organising code. In the models that were produced, the qualitative assessment showed that there were no obvious anti-patterns with respect to the functionality and classical coupling problems found in Object-Oriented are managed before they have a chance to take hold. In a way this is similar to a framework like Akka (Lightbend Inc, 2011), which removes many of the conditions that would produce race-conditions from occurring within parallel code.

- Inherent Scalability – The organisation of the code is believed to enable the ability to scale the functions onto parallel hardware as highlighted by Odersky (2014), but this has not been tested on GAMOV models thus far.

- Increased potential for reusability – Both the ADM and MCM models produced from the GAMOV framework have varying degrees of reuse. In the case of ADM, the model provides a basis from which to construct other logistics models and some of its mediators proved to be more independent that they could be applied to wider problems. For MCM this was less so, because the mediator combinations produced a framework of model that could only be reused on problems that are similar to MCM type optimisations. However, this is in stark contrast to older models where either no reuse was possible in the case of DIAMOND, or very limited to conceptual reuse of the ideas in COMAND.

With respect to the GAMOV implementation of Object-Functional, the following points should be noted:

- Interfacing – The organisation of code by splitting transformative functions away from the state that they manipulate allows for new ideas to be added and for pre-existing ideas to be amended or removed with a clear understanding of how the change will affect the other components in the code-base. This allows for an audit trail of changes to the model with respect to the validation of the representations. This will also enable discretised development practices and code-sharing strategies that could potentially be exploited on wider modelling frameworks such as the HLA.

- Single Methodological Ontology – It is believed that having all entities expressed with the same data structure helps to simplify the modelling. Whilst this is not a pre-requisite of Object-Functional, it could serve as a complimentary or extension pattern to Object-Functional for the organisation of state in software of this nature.

- Entity functions – Whilst the separation of functionality is at the heart of the paradigm, there are instances where it is permissible to still retain functionality within the state objects. This was highlighted with respect to bookkeeping functionality in the GAMOV entities within section [4.4.2.4]. In terms of GAMOV this is colloquial rule for the team, whereby functions can exist in the state objects provided they are private to the object (i.e. they don't manipulate state in other entity objects). This is an element of good practice that may wish consideration when framed in an Object-Functional design pattern, because at least in the defence context there are scenarios where state manipulation in this way is unavoidable.

6.8.3 Challenges of using Object-Functional

In addition to the benefits and observations made in the previous subsection, there are some points that are worth considering when using Object-Functional in order to mitigate potential problems.

- Component Levels – There needs to be some distinction between component levels in order to make it clear what is permissible in terms of interfacing. This was noted in the analysis of MCM, where it became clear that only certain types of mediators would be permissible within certain types of model. However, there is nothing in terms of code constraints to preclude a developer from

plugging those components together all the same, which could affect overall validation of their solution. Management of entity layers is something that would have to be managed externally with care.

- Protecting the Integrity of the paradigm – As noted in the overview of GAMOV, specifically, protection of the Object-Functional organisation is important. It is very easy for a subtle misinterpretation by a developer to violate the approach or incur technical debt if there is no formal and explicit guidance on how the paradigm is applied to the software. This should also apply higher up in terms of communicating the approach to other developers who may not be within the sphere of influence of the owning development team.

6.9 Chapter Summary

Within this chapter the overall learning with respect to Object-Functional has been collated and analysed from case studies of extant models and models built using the GAMOV approach. The next chapter shall present the conclusions to this research based upon this learning.

7 Conclusions

7.1 Chapter Summary

Within this chapter, the conclusions based upon all of the learning uncovered throughout this thesis shall be presented in support of the research questions. This shall include the insights gained from the evaluation of models built using other methods and paradigms versus those using Object-Functional under the GAMOV framework approach.

In addition to the formal evidence uncovered from evaluating the models, a number of lessons learnt shall also be covered as part of evaluating the overall experiences of developing GAMOV. Whilst these are empirical observations from a bespoke development and not a formal experiment, it is believed these insights are worth highlighting due to the evolving nature of the paradigm and the lack of formal declaration of its use in the literature. Even though these issues may not be experienced or even refuted later by similar developments, they serve as points worth considering for those venturing into this territory.

Finally, some of the future plans for GAMOV development shall be presented, as well as other related research avenues that I briefly explored as part of this research, but remain unaddressed.

7.2 Overall conclusions in support of the research questions

7.2.1 What benefits does the Object-Functional paradigm bring to defence modelling?

It is believed that there are a number of demonstrated benefits brought about by exploiting the Object-Functional paradigm, compared to previous development examples. The benefits that were demonstrated in support of this research question include:

- The ability to easily add new functionality to a model through the use of the mediator interface. The mediator provides a de-coupled mechanism with a defined interface that allows functionality to be added and used by any entities that have the requisite data values. This provides an iterative approach to model development, where functionality can be added, adapted or even

removed. This also strengthens the ability to perform verification and validation, because the impact can be measured from making these iterative changes to the model.

- For both the ADM and MCM models that was constructed using the GAMOV object-functional approach, some benefit of the reuse potential of the framework exists. Because there already exists a reusable time management system, configuration management system and the structures required for the entities and mediators within the GAMOV framework, the modeller is not required to re-invent the wheel each time a model was built. Compared to the old suite of campaign models, where these components are not inherently modular, these were constructed from scratch each time. Whilst the full potential of this aspect will only emerge with more model developments, the fact that we could effectively launch straight into designing and implementing model functionality and focus on building a representation through entities is believed to be a much more agile environment. It could be argued from a high-level perspective that the DROMAS framework provided similar capability; however, DROMAS was far harder to adapt due to the level of coupling between the framework components and the resultant model merely being a parametrisation of DROMAS. There was also significantly more overhead at the beginning of each model development due to the heavy focus on visualisation in that framework.

- All of the components within the GAMOV approach to Object-Functional do not have to be used solely for the purposes of GAMOV models. Each of the components are free of context with respect to the overall framework they sit in and only form something that is reflective of a GAMOV model when used together. Due to the decoupling brought about by using functional classes, each component can be used in isolation to support other model developments within Dstl. For example, the random number generator within GAMOV is an off-the-shelf component that could underpin any model requiring stochastic elements. Another example would be the configuration management system for creating parametric variations. This is a common task for most analysis studies requiring a model and is an element that is continuously re-built for each development.

- This suggests that GAMOV may need to branch out beyond its original remit in order to be more akin to a warehouse type approach, whereby modellers can access functionality in a more off-the-shelf manner.

There are also some potential benefits that the Object-Functional approach enables; however, these have not necessarily been demonstrated in practice with the model developments to date. These include:

- Only a limited amount of model re-use has been achieved so far with the approach. A goal of creating the GAMOV framework and exploiting Object-Functional was to reduce the amount of new development effort each time a model was required, by repurposing pre-existing models for a new study. The full potential of this ambition can only be demonstrated in time and continued usage; however, the potential for this is believed to exist. Going forward the amount of component reuse being experienced with the GAMOV framework should be monitored, in order to claim that benefit is being gained in terms of time and associated cost compared to historic model developments.

- As a result of using a single data structure approach for the representation of entities, the amount of code in the GAMOV code-base is believed to be lean, when compared to our previous models. The separation of functionality also aids in our ability to diagnose problems and bugs in the framework quickly. However, it would difficult to prove claim with any certainty at present because our extant models, also suffer from significant problems inherent from their development histories that are not necessarily a product of the paradigm that they were using. All the same, this would be an interesting benefit to explore and further assess in particular whether the Object-Functional paradigm does in-fact produce better code (and in what sense) comparatively to Object-Orientation.

7.2.2 What challenges does the Object-Functional paradigm bring to defence modelling?

Whilst appreciating all of the benefits listed above and, in the analysis, there are some potential challenges Object-Functional poses:

- As stated in the literature by Sousa & Ferreira (2012), there is a significant lack of design patterns in existence for the Object-Functional paradigm. The

GAMOV framework presents a high-level example of a design pattern with respect to the entity and mediator structures. However, because there lacks an explicit representation of this approach within the literature, it can make it difficult for new software developers to understand the entity-mediator relationship simply from the code-base alone, because they have nothing to compare it to. This could place the integrity of paradigm at risk within the software unless it is tightly controlled. For example, as stated in the overview of GAMOV there are no hard limitations in the mediator concept to stop a developer from breaching it and produce a subtle variation that could produce internal coupling within the models.

- As shown in the analysis of ADM; that model was shown to be using an outdated version of the core entity template used to build GAMOV models. Comparatively to MCM, this could mean that ADM is operating on outdated characterisations of entities or is not making use of more generic functionality that had been refactored into the entity with later model developments. Not only does this produce technical debt, it provides an overhead that needs to be managed across all models produced from the framework. This is an overhead of GAMOV more so than Object-Functional itself.

- Whilst Object-Functional is believed to provide increased flexibility, there is an onus on the developer to manage that flexibility so that it is not misused. Object-Functional allows new ideas to be added and extant ideas to reused, but there are very little rules about what can and cannot be used in conjunction with one another. In other words, Object-Functional provides a good framework for experimenting and progressively realising a valid model, but does provide any built-in guarantees. Going forward, there is a requirement to manage mediators in layers (similar to REST) so that there are at least implicit rules of what can interoperate.

7.3 Other lessons learnt from development

7.3.1 Protecting the integrity of the Object-Functional paradigm

Throughout the development of GAMOV, the team progressively acknowledged the importance of protecting and maintaining the integrity of the Object-Functional architecture of the GAMOV framework. However, when either introducing new

developers to the GAMOV team, or when interacting with potential customers representatives for the first time, we sometimes encountered scenarios where the Object-Functional structure of GAMOV could have been compromised without careful control and education by the development team. A key example of this occurred during the design of the MCM model. It was assumed by all parties that the customer representative for the MCM model understood the implementation of the Object-Functional approach. However, it was evident from the first pass at generating the user requirements for MCM that whilst there was clear understanding of the essence what the approach is trying to achieve, there was a lack of transparency regarding the concept of instantiating new entity capabilities as functional classes. Thus, the requirements for new functionality within the user requirements documentation was framed purely in terms of changes that were required to the entity structures (both in terms of additional attributes and functionality). There is a danger for non-developers, who are used to thinking in terms of how the old models are developed, not appreciating the subtle differences in the Object-Functional approach versus Object-Orientation. Small misunderstandings of this nature could have drastic consequences for the integrity of the framework, requiring diligence from the developers maintaining frameworks that use this paradigm.

7.3.2 Model design time

One aspect that frankly surprised the GAMOV development team was the amount of design time inherent with either adding a new component to the framework or in the construction of a model. Whilst it is believed that GAMOV has increased agility of modelling in terms of implementation effort, it does not (and arguably should not) decrease the amount of conceptual effort involved in designing a model. In fact, we found particularly in the development of MCM that the potential savings gained from providing agility to the implementation have now been taken up by putting more effort into designing components and understanding the implications of plugging components together. More usage and the application of metrics on studies would be required in order to substantiate this claim.

This raises an interesting question about whether agility in the implementation has provided any savings that can be exploited for the later analysis from models, if it is now being consumed in the design process. However, extra time spent during the design to ensure that components are fit for purpose, free of technical debt etc. may

in fact provide that extra confidence in the final product and give the analysts flexibility to look at and assess more cases. This can only truly be answered by extended use of the approach and monitoring with metrics.

Hypothetically speaking, even if the GAMOV framework contained every possible capability and mediator required to build any model; a good amount of time still needs to be spent considering the validation implications of combining those elements. Adapting a pre-existing model will reduce this to a certain extent, as the modeller would have a validation record from which to measure the impact of their changes. However, this is not the same for models being built from scratch. Additionally, the subject matter knowledge required to understand the impact is not always explicit within the model. Aspects such as conflicts arising from levels of data aggregation can be inferred from the entity descriptions, but the implications upon the high-level analysis cannot.

The application of semantics using a tool such as SWEEP (which shall be outlined within section [7.4.2] later), could help in this regard, as this would provide a mechanism for this information to be recorded within the model files. This would enable an analyst who does not possess all of the subject matter knowledge to at least be prompted as to the potential meaning of the results they are seeing.

7.3.3 Articulating Object-Functional to other developers and non-developers

A continuous problem we experienced throughout the development of GAMOV was communication of the concept to people outside of the team. This was not just in terms of articulating the benefits of the capability, but explaining what the capability is (and more importantly, what it is not) at a fundamental level to non-technical people.

It was found that the existence of a working model was required in order to teach new developers how to use the approach, because through observing a model in execution provided the clarity necessary to understand the entity-mediator relationship. Within a classical OO approach, it may be possible for a new developer to infer more from the static code-base because the context of use is arguably clearer. Objects have names, attributes and services that can define their purpose in the context of the overall program. In an Object-Functional program the functions are independent and stateless, meaning that whilst a developer can see what they do, they can potentially lack context in relation to the rest of the program. The entity

structure, whilst extremely flexible and preferable for our purposes in comparison to previous campaign models, also compounds this problem with understanding further, because the entity structure within the code-base communicates no meaning to the developer unless you can observe what it is doing in the context of the model and what functionality it is using. In the absence of formal software design patterns, a working model is crucial in order to communicate these concepts to new developers.

It was found that by directing new developers towards the REST architecture (prior to the acknowledgement of Object-Functional) significantly helped in bridging the gap in understanding that functions are stateless with uniformed interfaces and they serve the requirements of the clients, which in GAMOV's case are the entities.

7.3.4 Implementation Tools

Upon reflection the application of Python to developing an Object-Functional approach is believed to be a good decision. The language provided enough functional programming capabilities in order to practice the Object-Functional design, whilst making the final product elegant and explicit enough for new developers to understand. This was experienced numerous times when developers were circulated into the team, each of which were able to get up and running with development in less than a week. For a capability such as COMAND, from past experiences by the custodian this would typically take a number of weeks to become competent.

However, it would be useful to question whether a pure Python[3] implementation was appropriate for all aspects of code-base. During the development of MCM it was noted that the implementation of mission planner optimisers might have been better in a language that was more scalable. This could have been in either another implementation of Python, such as Jython[4] or another language entirely such as

[3] Not to be confused with the colloquially known 'Pure Python' phrase used by the Python community, which means building capabilities that can be executed solely by the Python interpreter.

[4] Jython is a Java Virtual Machine implementation of the Python language. The language is not constrained by a Global Interpreter Lock found within classic Python, making it more suited for parallel processing.

Scala (Odersky, 2014). However, the status quo of our infrastructure precluded us from addressing this during the development.

Going forward, expanding the pool of languages used for the development of mediators should be explored, so that the most appropriate tool for the job is being used as opposed to a single tool for all jobs. This could be achieved using a Remote Procedure Call (RPC) framework, such as Apache Thrift (Slee, Agarwal & Kwiatkowski, 2007). The Object-Functional nature of GAMOV makes the exploitation of this approach a quick win, because the mediators would form the 'servers' expected under RPC architectures and the GAMOV engine would form the 'client' calling those servers. Also given the cross-language support of Apache Thrift, this would enable mediators to be written in languages most suited to their requirements, whilst retaining the ability to seamlessly integrate with the components specific to the GAMOV approach.

Whilst this approach would increase the number of languages being utilised within the GAMOV capability, it would enable us to better integrate pre-existing algorithms into the framework, without needing to re-implement the code into Python. However, it is believed that in practice this would be a rare occurrence, given that the sources to draw upon would highly-coupled models, such as COMAND. However, on occasion, models written in other languages have been identified, where the functions are suitably interfaced that they could simply slot into the Object-Functional organisation of GAMOV using an RPC approach without significant re-implementation effort.

7.4 Future Plans

So far, this chapter has outlined the conclusions and lessons learnt from the experiences of developing GAMOV and applying the Object-Functional paradigm. This last section shall outline some of the future plans currently being considered, including one research avenue to improve the analysis of GAMOV models, which was briefly considered during the course of this research, but not pursued further.

7.4.1 HPC Resources

An important aspect that the team was always mindful of when constructing GAMOV was that in the future we would need to access HPC or super computing resources to

both speed up the run-time and improve the computational efficiency of some of the algorithms. Whilst this was not the main focus during the early stage of development, we ensured that the structure of GAMOV did not preclude us from exploiting these resources. It is believed that entity-mediator implementation of the Object-Functional approach natively allows GAMOV models to scale very easily to parallel based architectures, which is also the conclusion of Odersky (2014) with the application of Scala. Given that the functions are their own objects, this will enable us to relate the functions to their own processes and thus assign them to their own hardware resources very easily. Additionally, because mediators operate either independently, or structurally as a bounded subsystem, we can potentially assign them to processes knowing that there is very little to no dependence on the outputs of other processes, which could limit the potential for producing race conditions or deadlocks at the process level. This would at the very least give us a brute force speed up in terms of the runtime for various components.

The question then is whether threading the individual mediators in order to gain any potential efficiency from the inner workings of the algorithms themselves is of benefit. On the surface it is believed that this is something we may want to do on a case-by-case basis; however, the team would want to explore quicker-wins first to see if can improve the runtime of an algorithm before taking this route. An example of this would be the implementation of the mission planner optimisers in the MCM model. Our initial verification tests of the mission planner indicated to us that whilst there would be potential benefit to thread the optimiser used for generating the plans, much of the runtime was actually being caused by memory usage. In the case of the mission planner, we had actually generated a scenario where it was not clear to the automatic garbage collector within the Python language as to when a planning entity within the mission planner was no longer needed. As a result, the mission planner was using vastly more memory than was required and the subsequent read/write actions on that memory were actually causing a more significant delay than the genetic program used to optimise the solution. Once the team had identified this, we were able to bring the runtime of the mission planner down by a factor of 8, by writing new routines to aid in the releasing of that memory. Whilst there may be worth in threading the solver, this example illustrated to the team that threading should not be done as a matter of course or in the first instance in the case of runtime problems associated with GAMOV models. It is worth taking the time to profile and discover if there are

quicker, higher impact, wins to bring the runtime down into acceptable time frames, rather than taking the already constrained development time in order to squeeze out efficiencies at the thread level, unless the algorithm is naturally scalable.

7.4.2 Providing Agility to Analysis

For the entirety of this document, the focus has been on improving the agility of the modelling process in terms of implementation with a view to allowing more time and resources to undertake analysis. The roots of this research originally began in attempting to understand the associated challenges with respect to the agility of analysing the outputs from a model. The understanding and the importance behind the Object-Functional paradigm were drawn out as an associated enabler for this particular problem area and were subsequently worked up in more detail for this thesis through the development of GAMOV when they were identified during the late stages.

However, a good deal of thinking was also put into conceptualising a potential solution for analysis of outputs, in the form of the Semantic Web Examiner of Emergent Phenomena (SWEEP) (Toomey, 2016). The SWEEP tool was envisaged to be a companion tool to work in cooperation with (or as part of) the GAMOV framework. Where the GAMOV framework would speed up the implementation of models, SWEEP was envisaged to assist the analyst in processing the outputs of those models. The long-term goal of SWEEP (as per its namesake) was to capture and interpret emergent phenomena produced within the models. The fundamental problem with respect to emergence within our modelling is that there is no simple mechanism that can enable analysts to engage with it. This has been highlighted as key challenge for the fields of both operational research and OA (Sterman, 2000).

The key reason for this is that the nature of emergence is a product of more than the sum of its parts. In other words, an emergent effect is usually produced by either more than the variables being monitored or the unique circumstances by which they interact with one another (Georgiou, 2007). These may either be sufficient or insufficient to produce a state-change within the model, which shapes the flow of subsequent state changes within a simulation. Such emergent phenomena can impact a simulation many times, producing either positive or negative feedback loops that can lead to the collapse of the system (Johnson, 2006). It is the transitory but

pivotal nature of emergence that shapes our understanding of the possibilities within the outcome space of a simulation (Georgiou, 2007). A lack of emergence within a simulation implies that all aspects of representation are understood, and thus no further learning can be drawn from the simulation. Therefore, emergence is a fundamental requirement in order to draw new insights from a constructive simulation, but there is no clear means to engage with it. This means that key potential insight of significant use to the MOD from models is not at present being exploited. For the analysis component of OA, this means that the level of understanding, which is drawn from modelling, is much less than it could be. This is simultaneously a value for money issue, and a quality requirement, which leads to lost opportunities to better shape and inform the requirements of MOD.

A common conceptual model for emergence, used often in the field of ecology is the idea of scalar hierarchies built upon experimental frames (O'Neill, DeAngelis, Waide, & Allen, 1986). There are effectively three levels (A, B and C) providing a descriptive framework and the context for which emergent behaviours can arise.

Frame A is considered the lowest level, with frame C being the lowest; with higher-level frames containing the lower level frames. Emergent properties are seen as: *"Representing something new at a given level that is not seen at the level below"* (Aumman, 2007).

In OA terms, Frame A is where our Measures of Performance (MoP) reside and is where the constituent parts of the higher-level entities or constructs are described (i.e. attributes in Object-Orientation or Object-Functional). Frame B is where the Measures of Effectiveness (MoE) are defined.

The behaviours observed within Frame B are *"new qualities that appear on higher integration levels… represent more than the sum of the low-level components"* (Reuter et al., 2005). These behaviours observed within Frame B are referred to as 'emergent'.

Taking an example from Ecology (Aumman, 2007), if you want to measure the growth rate of a crab, you cannot interpret all of its behaviours from the MoP's, such as the rate at which it is capable of feeding and its energy usage. An additional factor not captured within Frame A in determining the growth rate is the availability of food along the crab's movement path. This is a factor that cannot be subject to a priori

measurement since the movement path depends upon the crab's response to wider environmental factors that can only be understood through running a simulation. Therefore, the growth rate of the crab is considered an emergent behaviour consisting of components more than those that are defined within Frame A. Frame C is where the Measures of Outcome (MoO) are generated as a result of more detailed interactions occurring within Frames A and B, which together describe the crabs within their environment.

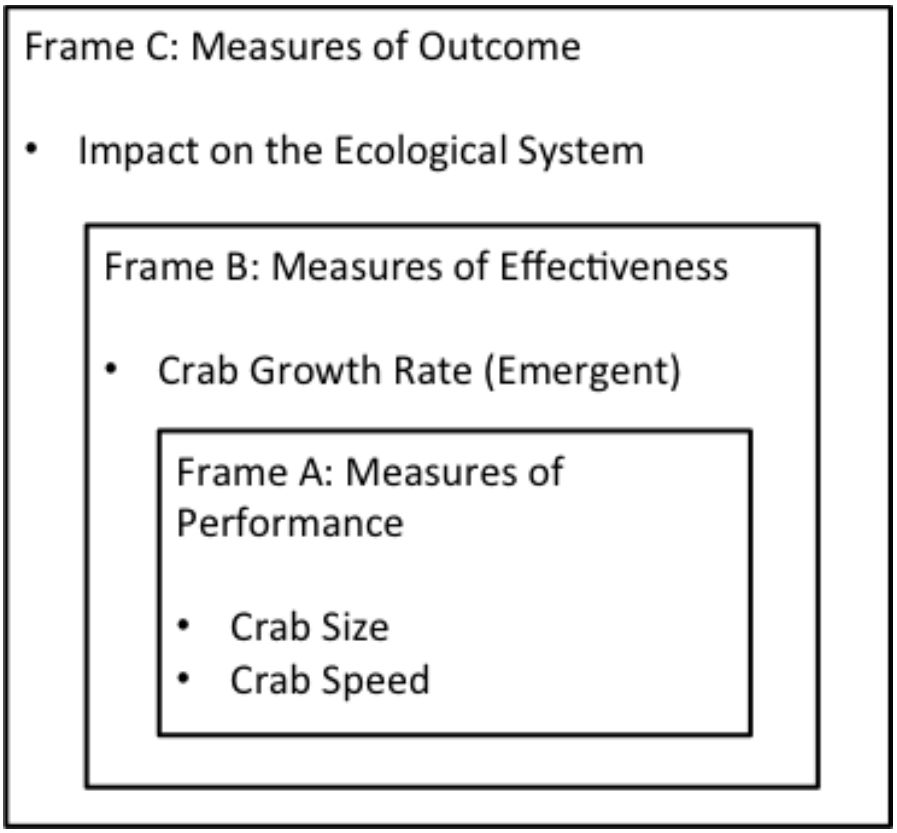

Figure [9]: Descriptive Framework of Emergence.

The general concepts of this method can usefully be read across to the defence context and campaign models, specifically GAMOV. Frame A would constitute the attributes of the entities and other forms of data including orders and missions within plans. The contents of Frame B are the constituent entities themselves, the flow of information across the simulation and elements such as command and control hierarchies. Frame C, the measure of outcome would be whether the parts that make up frames A and B achieve the outcome of the campaign. The key differences between ecological and defence modelling can be simply explained through the

consideration of Perrow's quadrants (Czerwinski, 1998). In the case of ecology there is a strong inter-linkage between the production of properties in a lower frame and the resultant changes in the higher frame, since even though irreducible to the sum of the parts the emergent characteristics are nonetheless part of a story of on-going adaption with broad periods of stability. Perrow would describe such a system as being tightly coupled[5] due to the tight inter-linkage between production of properties and resultant change. In contrast Perrow would describe defence as being loosely coupled, since the modelling actively examines competing systems, where each system has the potential to dominate the situation. Defence modelling contains antagonistic interactions, whereas many other fields operate on the notion of synergistic interactions. This is what makes analysing defence situations fundamentally difficult.

Prior to an emergent circumstance being realised there are indicators that can be measured in the lower frames. This has long been understood within the defence intelligence communities, whereby identifying these key indicators allows for the construction of unfolding situations, which will allow security forces to understand how, and when something is going to occur. The big idea here is being able to characterise and apply the essence of these processes to the analysis of simulations. Consider for example an observable event, such as the destruction of a military vehicle or the failure of a mission. From an analysis perspective it would be desirable to recognise and describe the competing processes that surround such events, especially if they are very much game-changing moments within a campaign outcome. Having an automated means to identify not only those things that happened, but also those things that almost happened but did not, would be a game-changer for the overall capability of defence modelling.

Another example of emergent behaviour within defence modelling is the concept of 'breakthrough', which results from the combination of the momentum of an attacking unit and the lack of ability on the part of the defensive unit to organise adequate resources to oppose the attack. Representing all of the components of breakthrough phenomena within a model enables a valid representation of the effect to be represented explicitly. By contrast, a simple model of breakthrough derived from an

[5] Not to be confused directly with the notion of coupling as already outlined within this document. The notion here is the coupling of ideas in terms of data and interactions.

empirical basis would include the effect of the complexity but not the means by which to engage with the complexity (Chialvo, 2008). In summary the key to generating emergence also comes by having the right resolution within the representation, otherwise you will lack the ability to extract even with access to an automated means.

The journey towards such a solution is believed to exist in the exploitation of Semantic Web technologies, built upon some of the key ideas presented by Boakes (2007). The Semantic Web technology stack provide the mechanisms by which the data and functionality of our models could be encoded in order to record the flows of information throughout the models and the key state changes that lead to the production of emergent phenomena. It is believed that the GAMOV framework will provide both effective platforms to allow for continued research into this area and that the Object-Functional architecture of the code is key to providing the explicit inter-linkages between the state and the functions that are changing that state. From the literature concerning emergence as outlined in this section, combined with the understanding within this thesis, it is believed that Object-Functional provides a best of both worlds' solution. The separation of state and transformative functions aids in reducing the coupling that causes such significant implementation and maintenance issues for the models, whilst leaving the explicit linkages that are key to grappling with generation of emergent phenomena. Having these explicit linkages is key, which is why a solution has been found with respect to system dynamics (Mojtahedzadeh, 1997). For a computer-driven simulation, the automated means of achieving this is the concept of SWEEP, which shall be a future research direction building on top of GAMOV. It is hoped that SWEEP shall be an iterative process (akin to model-test-model) whereby the modeller can encode their perspectives within the model, test the outcome and progressively develop their ontological view of the world in order to extract understanding from the model.

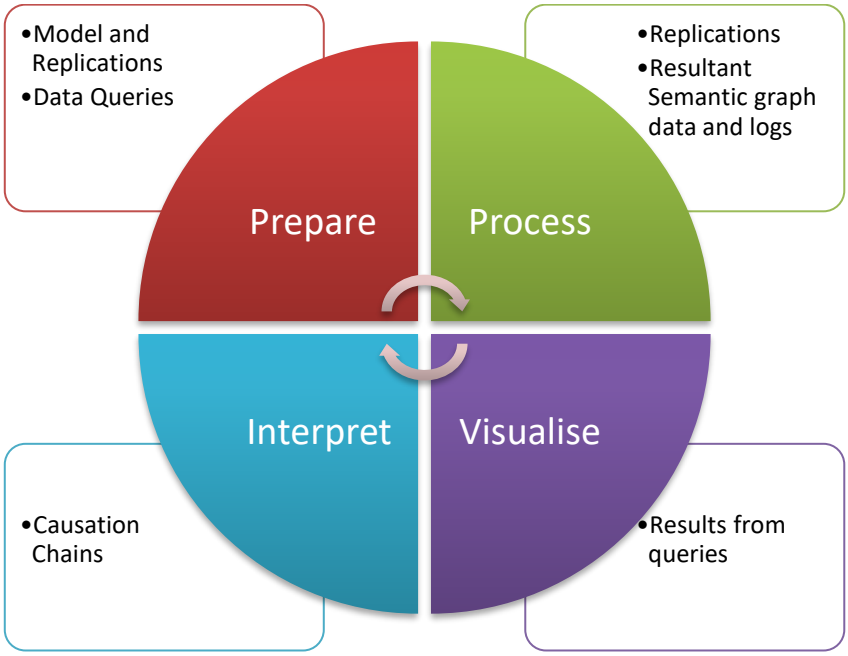

Figure [10]: The SWEEP Lifecycle

The application of semantic web-based approaches has previously been attempted within modelling, as outlined by Hoffman, Palli and Mihelcic (2011). This work emphasises the importance of having a consistent ontological view of the world across the model, as well as representing the situation in as much detail as possible. It is believed that the GAMOV entity structure will help in this regard as all entities are described using a single methodological ontology.

It is envisaged that through extensive research and development over a number of years in conjunction with the developed usage of GAMOV, a SWEEP type capability could aid in a number of key areas, such as:

- Helping an analyst to quickly distinguish between errors of validity vs. areas of analytical interest. Both of these are capable of providing fundamental insights for different reasons; however, the current methods require manual processing by the analyst, which is extremely time consuming. As a result, only a handful of potential problems can be processed within the timeframes of a typical study and there is not necessarily any guarantee that the effort is being targeted in the right areas to begin with. Having the ability for an analyst to identify where best to focus their efforts with their limited analysis time would enable better allocation of resources.

- Errors in validity can reveal understanding in how the model is being used (either correctly or incorrectly[6]) or potential problems with the representation.

- These are the areas where potentially new insights can be derived about the representation (i.e. the presence of emergence). Many of these are potentially lost through a lack of their discovery due to the available time, which limits our ability to provide key insights to MOD.

- Providing more comprehensive outputs to our customers. In the majority of cases we can only explain what has happened. We cannot necessarily identify the circumstances that were present (at a point in time) to produce the observable outcome; if we can even pinpoint the exact time when the production of the effect took place, we can begin to build a picture of the circumstances that led to its production.

[6] The model can be used correctly but in doing so reveals a feature or implementation error. The inverse can be true indicating a poor approach to usage by the modeller.

Bibliography

Abbes, M., Khomh, F., Gueheneuc, Y-G. (2011). An Empirical Study of the Impact of Two Anitpatterns, Blob and Spaghetti Code, on Program Comprehension. *Proceedings to the 2011 15th European Conference on Software Maintenance and Reengineering. 181 – 190.* https://doi.org/10.1109/CSMR.2011.24

Ahuja, S., Gelernter, D. & Carriero, N. (1986). Linda and Friends. *Computer, 19(8).* https://doi.org/10.1109/MC.1986.1663305

Ansell, R., & Glover, P. (2008). *The Software approach to be used for GAMOV.* Unpublished internal document. Defence, Science & Technology Laboratory (Dstl).

Aumman, C. (2007). A Methodology for developing simulation models of complex systems. *Ecological Modelling. 202(3-4), 385-396.* https://doi.org/10.1016/j.ecolmodel.2006.11.005

Boakes, R. J. & Hanley, N. (2017). *GAMOV Technical Deep-dive assessment notes.* Unpublished Dstl Internal Report.

Boakes, R. J., & Toomey, G. (2012). Developing Parallelised Modelling Environments using GAMOV [Abstract]. *Proceedings of the 54th Annual Conference on Operational Research (OR54).*

Boakes, R. J. (2007). *Semantic Web-Based Log analysis for distributed systems and application.* Ph.D. Thesis. University of Portsmouth.

Brooks Jr., F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition.* (2nd Edition). Boston U.S.: Addison-Wesley.

Buss, A (2002). Component based simulation modelling with Simkit. *Proceedings of the 2002 winter simulation conference. 243-249.* https://doi.org/10.1109/WSC.2002.1172891

Chialvo, D. (2008). Emergent Complexity: What uphill or downhill invention cannot do. *New Ideas in Psychology. 26(2), 158-173.* https://doi.org/10.1016/j.newideapsych.2007.07.013

Clive, P. D., Johnson, J. A., Moss, M. J., Zeh, J. M., Birkmire, B. M., & Hodson, D. D. (2015) Advanced Framework for Simulation Integration and Modelling (AFSIM). *International Conference Scientific Computing CSC'15. 73-77.*

Czerwinski, T. (1998). *Coping with Bounds, A Neo-Clausewitzean Primer*. CCRP

Defence, Science & Technology Laboratory. (2015). *Defence, Science and Technology Laboratory Overview.* Retrieved from: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/641731/Generic_factsheet_v2.pdf

Dhal, O. J. (2002). *The Roots of Object Orientation: The Simula Language.* In Broy, M., & Denert, E. (Eds.), Software Pioneers. (pp. 78-90). Berlin-Heidelberg: Springer-Verlag.

Dahmann, J. S. (1997) High Level Architecture for Simulation. *Proceedings First International Workshop on Distributed Interactive Simulation and Real Time Applications. 9-14.* https://doi.org/10.1109/IDSRTA.1997.568652

Drobi, S. (2007). *OOP: Thinking beyond verb/noun metaphor to yield a better design.* Retrieved from: https://www.infoq.com/news/2007/11/oop-beyond-verb-noun

Editors of the Encyclopedia Britannica. (n.d.). *George Gamow Biography.* Retrieved from: https://www.britannica.com/biography/George-Gamow

Erikson, M. (2012). *Effective Scala. [Online GIT repository]* Retrieved from: https://twitter.github.io/scala_school/

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* (PhD Dissertation). University of California, Irvine. Retrieved from: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Fletcher, D. J., Lukman, N. B., & Hodson, M. J. (2005). Principals of simulation architecture-independent model development [Abstract] *SimTecT Conference Sydney, -12 May 2005*.

Fontana, F. A., Zanoni, M., Marino, A., & Mäntylä, M. V. (2013) Code Smell Detection: Towards a Machine Learning-Based Approach. *2013 IEEE International Conference on Software Maintenance. 396-399.* https://doi.org/10.1109/ISCM.2013.56

Fujimoto, R. M. (n.d.) *The High-Level Architecture Introduction.* Retrieved from: https://www.acm-sigsim-mskr.org/Courseware/Fujimoto/Slides/FujimotoSlides-20-HighLevelArchitectureIntro.pdf

Glover, P., Collander-Brown, S., & Taylor, S. J. E. (2017) Using A Genetic Programming Approach to Mission Planning to Deliver More agile Campaign Level modelling for Military Operational Research. *Proceedings of the 2017 Winter Simulation Conference, 4465 – 4467.* https://doi.org/10.1109/WSC.2017.8248165

Glover, P., & Toomey, G. (2012). GAMOV: An Agile Generic Simulation Tool for Military Joint Forces Modelling. *Proceedings of the Operational Research Society Simulation Workshop 2012 (SW12), 275-279.* Retrieved from: http://www.theorsociety.com/Pages/ImagesAndDocuments/documents/Conferences/SW12/Papers/GloverToomey.pdf

Georgiou, I. (2007). *Thinking through System Thinking.* Abingdon, Oxon: Routledge

Harrison, R., Samaraweera, L. G., Dobie, M. R., & Lewis, P. H. (1996). Comparing programming paradigms: an evaluation of functional and object-oriented programs. *Software Engineering Journal, 11(4), 247-254*. http://doi.org/10.1049/sej.1996.0030

Hoffman, M., Palii., J & Mihelcic., G. (2011). Epistemic and normative aspects of ontologies in modelling and simulation. *Journal of Simulation 5(3), 135-146*. http://doi.org/10.1057/jos.2011.13.

HM Treasury. (2015). *The Aqua Book: guidance on producing quality analysis for government.* Retrieved from: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/416478/aqua_book_final_web.pdf

HM Treasury. McPherson, N. (2013). *Review of Quality Assurance of Government Analytical Models: Final Report.* Retrieved from: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/206946/review_of_qa_of_govt_analytical_models_final_report_040313.pdf

IEEE Standards Association. (2010). *1516-2010 - IEEE Standard for Modelling and Simulation (M&S) High Level Architecture (HLA)-- Framework and Rules.* Retrieved from: https://standards.ieee.org/findstds/standard/1516-2010.html

International Organisation for Standardisation (ISO). (2011). *Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.* Retrieved from: https://www.iso.org/standard/35733.html

Jalote, P. (1989) Functional refinement and nested objects for object-oriented design. *IEEE Transactions on Software Engineering. 15(3). 264-270.* https://doi.org/10.1109/32.21754

Johnson, C. W. (2006). What are emergent properties and how do they affect the engineering of complex systems. *Reliability Engineering and System Safety. 91. 1475-1481.* https://doi.org/10.1016/j.ress.2006.01.008

Kester, J. E. (1993). Some Limitations of Object-Oriented Design. *IEEE Aerospace and Electronic Systems Magazine. 8(9), 14-16.* https://doi.org/10.1109/62.257112

King, D. W., Hodson, D. D., & Peterson, G. L. (2017) The role of simulation frameworks in relation to experiments. *Proceedings of the 2017 Winter Simulation Conference. 4153-4161.* https://doi.org/10.1109/WSC.2017.8248123

Kontio, M., Mäyrä, H., & RÖnkkÖ, M. (2007) Functional Classes Guide Use of Design Patterns in Implementing Mediators. *Proceedings of the First International Conference on Complex, Intelligent and Software Intensive Systems.* https://doi.org/10.1109/CISIS.2007.29

Kristensen, J. T., Hansen, M. R., & Rischel, H. (2001). Teaching object-oriented programming on top of functional programming. *31st Annual Frontiers in Education Conference, 2001.* https://doi.org/10.1109/FIE.2001.963848

Lau, M. (2015, August, 31). *The Essence of Object-Functional Programming and the Practical Potential of Scala.* Retrieved from: https://blog.codecentric.de/en/2015/08/essence-of-object-functional-programming-practical-potential-of-scala/

Lightbend Inc. (2011). *Akka.* Retrieved from: http://akka.io/

Lloyd, J., Newton, N., & Perkins, R. (2014) A Chemical, Biological and Radiological Modelling Capability to Support Acquisition Advice and Re-use as a Common Cross-Domain Capability. *North Atlantic Treaty Organization (NATO) Science and Technology Organization (STO) Meeting Proceedings.* https://doi.org/10.14339/STO-MP-MSG-126-09-pdf

Lucek, S., G., & Collander-Brown, S. (2014) Artificial intelligence algorithms and new approaches to wargame simulation. *Proceedings of the 31st International Symposium on Military OR.*

Mansfield, R. (2005). *Has OOP Failed?* Retrieved from: http://4js.com/files/documents/products/genero/WhitePaperHasOOPFailed.pdf

Matsumoto, M. (2007*). Mersenne Twister Home Page: A very fast random number generator of period 219937-1.* Retrieved from: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

McCullough, B, D. & Wilson, B (2005) On the accuracy of statistical procedures in Microsoft Excel 2003. *Computational Statistics and Data Analysis. 49(4), 1244 – 1252.* https://doi.org/10.1016/j.csda.2004.06.016

McManus, J., & Wood-Harper, T. (2008). *A Study in Project Failure.* Retrieved from: http://www.bcs.org/content/ConWebDoc/19584

Miguel, J, P., Mauricio, D. & Rodriguez, G. (2014) A Review of Software Quality Models for the Evaluation of Software Products. *International Journal of Software Engineering & Applications.* 5(6), 31 – 53. https://doi.org/10.5121/ijsea.2014.5603

Moffat, J. (2011). *Adapting modelling & simulation for Networked Enabled Operations.* CCRP

Moffat, J., Scales, T., Taylor, S., & Medhurst, J. (2011). Quantifying the need for force agility. *The International C2 Journal 5(1), 1-23.* Retrieved from: http://dodccrp.org/html4/journal_v5n1.html

Moffat, J., Campbell, I., & Glover, P. (2004). Validation of the mission-based approach to representing command and control in simulation models of conflict*. Journal of the Operational Research Society 55(4), 340-349.* https://doi.org/10.1057/palgrave.jors.2601662

Mojtahedzadeh, M. (1997). *A Path taken: Computer-Assisted heuristics for understanding dynamic systems.* (Unpublished Ph.D. Thesis). Rockefeller College of Public Affairs. University of Albany

Naval Postgraduate School. Buss, A. (2004). *Simkit Analysis Workbench for Rapid Construction of Modelling and Simulation Components.* Retrieved from: http://calhoun.nps.edu/handle/10945/37865

Nesfield, A. E. S. (1998). *DROMAS System Description.* Unpublished internal document. Defence, Science & Technology Laboratory (Dstl).

O'Neill, R.V., DeAngelis, D.L., Waide, J.B., & Allen, T.F.H. (1986). *A hierarchical concept of ecosystems*. New Jersey, USA: Princeton University Press.

Odersky, M., Altherr, P., Cremet, V., Emir, B., Maneth, S., Micheloud, S., Mihaylov, N., Schinz, M., Stenman, E., & Zenger, M. (2006). *An Overview of the Scala Programming Language. 2nd Edition.* Retrieved from: http://scala-lang.org/docu/files/ScalaOverview.pdf

Odersky, M. (2014). *Unifying Functional and Object-Oriented Programming with Scala.* Retrieved from: https://cacm.acm.org/magazines/2014/4/173220-unifying-functional-and-object-oriented-programming-with-scala/fulltext

Ottinger, T., & Langr, J. (2011). *Code Coupling: Reducing Dependency in your Code.* The Pragmatic Bookshelf. Retrieved from: https://pragprog.com/magazines/2011-01/code-coupling

Pankratius, V., Schmidt, F., Garreton, G. (2012). Combining Functional and Imperative Programming for Multicore Software: An Empirical Study Evaluating Scala and Java. *34th International Conference on Software Engineering (ICSE), 2012. 2-9 June 2012.* https://doi.org/10.1109/ICSE.2012.6227200

Peters, T. (2004). PEP20 – *The Zen of Python.* Retrieved from: https://www.python.org/dev/peps/pep-0020/

Petty, M., & Weisel, E. (2003) A formal basis for a theory of semantic composability. *Proceedings of the Spring Simulation Interoperability Workshop.*

Pidd, M. (1996). *Tools for thinking: Modelling in Management Science.* Chichester: Wiley.

Poulter., A. J. (2011). The implications for the use of high performance and parallel computing techniques in OA modelling. *Proceedings of the Operational Research Society Simulation Workshop 2012 (SW12)* The Operational Research Society: Birmingham, UK.

Qian, H., Fernandez, E. B., & Wu, J. (1995) A combined functional and object-oriented approach to software design. *Proceedings of First IEEE International Conference on Engineering of Complex Computer Systems. ICECCS'95.* https://doi.org/ 10.1109/ICECCS.1995.479323

Reuter, H., Holker, F., Middelhoff, U., Jopp, F., Eschenbach, C., & Breckling, B. (2005). The Concepts of Emegent and Collective Properties in Individual Based Models – Summary and Outlook of the Bornhoved case studies. *Ecological Modelling. 186(4), 489-501.* https://doi.org/10.1016/j.ecolmodel.2005.02.014

Robinson, A. P., & Glover, P. E. (2006). Recent developments in high level defence operational research. *OR48 Defence Stream Keynote*. The Operational Research Society: Birmingham.

Roy, S., Hermans, F., & Van Deursen, A. (2017) Spreadsheet testing in practice. *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER).* 20-24 Feb. 2017. 10.1109/SANER.2017.7884634

Rubberduck VBA. Guindon, M., & McClellan, C. (2014). *Rubberduck.* Retrieved from: http://rubberduckvba.com/

Salt, J. D. (2008). The seven habits of highly defective simulation projects. *Journal of Simulation. 2(3), 155-161.* https://doi.org/10.1057/jos.2008.7

Sargent, R. J. (2005). Verification and validation of simulation models. 37th Winter Simulation Conference: 130-143 ACM https://doi.org/10.1109/WSC.2005.1574246

Slee, M., Agarwal, A., & Kwiatkowski, M. (2007). *Thrift: Scalable Cross-Language Services Implementation.* Retrieved from: https://thrift.apache.org/static/files/thrift-20070401.pdf

Soares de Jesus, J. & Vieira de Melo, A. C. (2017). Technical Debt and the Software Project Characteristics. A Repository-Based Exploratory Analysis. *Proceedings of the IEEE 2017 19th Conference on Business Informatics (CBI).* https://doi.org/10.1109/CBI.2017.62

Sousa, T. B., & Ferreira, H. S. (2012). Object-Functional Patterns: Re-Thinking Development in a Post-Functional World. *2012 Eighth International Conference on the Quality of Information and Communications Technology, 348-352.* http://doi.org/10.1109/QUATIC.2012.43

Sterman., J. (2000). *Business Dynamics: Systems thinking and modelling for a complex world*. Irwin McGrawHill: London.

Svallfors, H. (2011). *Sard: An Object-Functional Programming Language.* (Ph.D Thesis) Ume˚a University. https://doi.org/ 10.1.1.220.9799

Taylor, B., & Lane, A. (2004). Development of a novel family of military campaign simulation models. *Journal of the Operational Research Society 55(4), 333-339.* https://10.1057/palgrave.jors.2601714

Teo, Y. M., Szabo, C. (2008) CODES: An Integrated Approach to Composable Modelling and Simulation. *Proceedings of the 41st Annual Simulation Symposium. 103-110.* https://doi.org/10.1109/ANSS-41.2008.24

Toomey, G. (2016) An Object-Functional Modelling Platform to Enable Semantic Web Analysis of Campaign Models. *Proceedings of the Operational Research Society Simulation Workshop 2016 (SW16). 175-180.*

Van Rossum, G. (2009, February 27). *First-Class Everything.* Retrieved from: http://python-history.blogspot.co.uk/2009/02/first-class-everything.html

Wampler, D., & Payne, A. (2014) *Programming Scala: Scalability = Functional Programming + Objects (2nd Ed).* O'Reilly.

Webster, B. F. (1995) *Pitfalls of Object-Oriented Development*. M&T Books.

Yegge, S. (2006, March 30). *Execution in the Kingdom of Nouns.* Retrieved from: https://steve-yegge.blogspot.co.uk/2006/03/execution-in-kingdom-of-nouns.html

List of publications originating from this research

Boakes, R. J., & Toomey, G. (2012). Developing Parallelised Modelling Environments using GAMOV [Abstract]. *Proceedings of the 54th Annual Conference on Operational Research (OR54).*

Collander-Brown, S., Byrne, M., Toomey, G., Glover, P. (2013). A U.K. Perspective on Campaign Level Constructive Simulation. *Proceedings of the 30th International Symposium on Military Operational Research.* Retrieved from: http://ismor.cds.cranfield.ac.uk/30th-symposium-2013

Glover, P., & Toomey, G. (2012). GAMOV: An Agile Generic Simulation Tool for Military Joint Forces Modelling. *Proceedings of the Operational Research Society Simulation Workshop 2012 (SW12), 275-279.* Retrieved from: http://www.theorsociety.com/Pages/ImagesAndDocuments/documents/Conferences/SW12/Papers/GloverToomey.pdf

Toomey, G. (2016). An Object-Functional Modelling Platform to Enable Semantic Web Analysis of Campaign Models. *Proceedings of the Operational Research Society Simulation Workshop 2016 (SW16). 175-180.*

List of abbreviations

ADM	Aerial Delivery Model
API	Application Programming Interface
CBR	Chemical, Biological & Radiological
CIO	Chief Information Officer
COMAND	C3 Oriented Model of Air and Naval Domains
CSS	Cascading Style Sheet
C3	Command, Control and Communications
C4	Command, Control, Communications and Computers
DERA	Defence & Evaluation Research Agency
DIAMOND	Diplomatic and Military Operations in a Non-war fighting Domain
DROMAS	DERA Reusable Object Modelling and Simulation
DSA	Defence & Security Analysis
DSTL	Defence, Science and Technology Laboratory
EPFL	Swiss Federal Institute of Technology in Lausanne
GAMOV	Generic Aggregator Model Valuator
GIL	Global Interpreter Lock
GUI	Graphical User Interface
HLA	High-Level Architecture
HLOA	High-Level Operational Analysis
HPC	High Performance Computing
ICT	Information Communication Technology
IDE	Integrated Development Environment
ISR	Intelligence, Surveillance and Recognition
JSON	JavaScript Object Notation
MCM	Mission Command Model
MFC	Microsoft Foundation Classes
MOE	Measure of Effectiveness
MOO	Measure of Outcome
MOP	Measure of Performance
LP	Linear Program
OA	Operation Analysis
ORBAT	Order of Battle
MOD	Ministry of Defence
REST	Representational State Transfer
RPC	Remote Procedure Call
S&T	Science and Technology
SLAM	Simple Land Air Model
SME	Subject Matter Expert
StratBOI	Strategic Balance of Investment
SWEEP	Semantic Web Examiner of Emergent Phenomena
VBA	Visual Basic for Applications
WPT	Wartime Planning Tool

APPENDIX A Ethics Documentation

Please see overleaf for captures of the original documentation.

FORM UPR16

Research Ethics Review Checklist

Please include this completed form as an appendix to your thesis (see the Research Degrees Operational Handbook for more information

Postgraduate Research Student (PGRS) Information				Student ID:	230731
PGRS Name:	Gareth Toomey				
Department:	School of Computing	First Supervisor:	Dr. Rich Boakes		
Start Date: (or progression date for Prof Doc students)	October 2010				
Study Mode and Route:	Part-time ☒	MPhil ☐	MD ☐		
	Full-time ☐	PhD ☒	Professional Doctorate ☐		

Title of Thesis:	An Application of Object-Functional Programming to Defence Modelling
Thesis Word Count: (excluding ancillary data)	37,202

If you are unsure about any of the following, please contact the local representative on your Faculty Ethics Committee for advice. Please note that it is your responsibility to follow the University's Ethics Policy and any relevant University, academic or professional guidelines in the conduct of your study

Although the Ethics Committee may have given your study a favourable opinion, the final responsibility for the ethical conduct of this work lies with the researcher(s).

UKRIO Finished Research Checklist:
(If you would like to know more about the checklist, please see your Faculty or Departmental Ethics Committee rep or see the online version of the full checklist at: http://www.ukrio.org/what-we-do/code-of-practice-for-research/)

a) Have all of your research and findings been reported accurately, honestly and within a reasonable time frame?	YES ☒ NO ☐
b) Have all contributions to knowledge been acknowledged?	YES ☒ NO ☐
c) Have you complied with all agreements relating to intellectual property, publication and authorship?	YES ☒ NO ☐
d) Has your research data been retained in a secure and accessible form and will it remain so for the required duration?	YES ☒ NO ☐
e) Does your research comply with all legal, ethical, and contractual requirements?	YES ☒ NO ☐

Candidate Statement:

I have considered the ethical dimensions of the above named research project, and have successfully obtained the necessary ethical approval(s)

Ethical review number(s) from Faculty Ethics Committee (or from NRES/SCREC):	7FFB-9D0A-8C83-2B44-D1BD-0D02-A512-E1D8

If you have *not* submitted your work for ethical review, and/or you have answered 'No' to one or more of questions a) to e), please explain below why this is so:

Signed *(PGRS)*:		**Date:** 13/05/19

University of Portsmouth

Certificate of Ethics Review

Project Title:	GAMOV - An Object-Functional Modelling Framework for Defence Operational Analysis
User ID:	230731
Name:	gareth david toomey
Application Date:	02/09/2017 14:54:31

You must download your certificate, print a copy and keep it as a record of this review.

It is your responsibility to adhere to the University Ethics Policy and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers and University Health and Safety Policy.

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

You are reminded that as a University of Portsmouth Researcher you are bound by the UKRIO Code of Practice for Research; any breach of this code could lead to action being taken following the University's Procedure for the Investigation of Allegations of Misconduct in Research.

Any changes in the answers to the questions reflecting the design, management or conduct of the research over the course of the project must be notified to the Faculty Ethics Committee. **Any changes that affect the answers given in the questionnaire, not reported to the Faculty Ethics Committee, will invalidate this certificate.**

This ethical review should not be used to infer any comment on the academic merits or methodology of the project. If you have not already done so, you are advised to develop a clear protocol/proposal and ensure that it is independently reviewed by peers or others of appropriate standing. A favourable ethical opinion should not be perceived as permission to proceed with the research; there might be other matters of governance which require further consideration including the agreement of any organisation hosting the research.

GovernanceChecklist
A1-BriefDescriptionOfProject: This project is examining the implications of exploiting the Object-Functional software paradigm in the development of modelling software. The customer for this research is the Defence, Science and Technology Laboratory (Dstl), who are part of the Ministry of Defence. The thesis will examine the usage of the OF paradigm in the creation of a new modelling framework within Dstl called GAMOV, which is seeking to improve upon extant implementations and bring greater agility to the implementation of models. Whilst the context of this work is defence oriented, no governmental data pertaining to persons, military personnel or real-world military

activities is being recorded or published in the thesis. The author is also an employee of Dstl and thus the work shall be subjected to formal review by Dstl who will provide authorisation for its publication, thereby handling any concerns with respect to security and reputation.
A2-Faculty: Technology
A3-VoluntarilyReferToFEC: No
A5-AlreadyExternallyReviewed: No
B1-HumanParticipants: No
HumanParticipantsDefinition
B2-HumanParticipantsConfirmation: Yes
C6-SafetyRisksBeyondAssessment: No
D2-PhysicalEcologicalDamage: No
D4-HistoricalOrCulturalDamage: No
E1-ContentiousOrIllegal: No
E2-SociallySensitiveIssues: No
F1-InvolvesAnimals: No
F2-HarmfulToThirdParties: No
G1-ConfirmReadEthicsPolicy: Confirmed
G2-ConfirmReadUKRIOCodeOfPractice: Confirmed
G3-ConfirmReadConcordatToSupportResearchIntegrity: Confirmed
G4-ConfirmedCorrectInformation: Confirmed